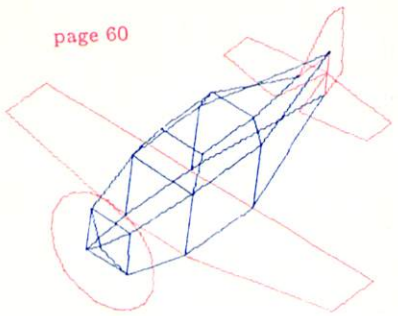
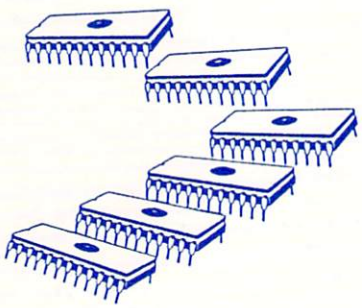
COMAL TODAY 13



3D PROJECTIONS

SUPER CHIP$_{tm}$ ARRIVES page 62

COMAL Today
6041 Monona Drive
Madison, WI 53716



Bulk Rate
U.S.Postage
Paid
Madison, WI
Permit 2981

IF YOUR LABEL SAYS
LAST ISSUE: 13
YOU MUST RENEW NOW.
USE ORDER FORM INSIDE

COMAL Today Issue 13 June 27, 1986

PUBLISHER
COMAL Users Group,
U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

EDITOR
Len Lindsay

ASSISTANTS
Richard Bain
Maria Lindsay
David Stidolph
Geoffrey Turney

ART
G Raymond Eddy

CONTRIBUTORS
Richard Aurland
Phyrne Bacon
Richard Bain
Jack Baldridge
Ed Bolton
Reed Brown
Doug Colpitts
Captain COMAL
Bert Denaci
Michael Erskine
Robert Gerber
Oren Hasson
Gerald Hobart
Bob Hoerter
Daniel Horowitz
Dan Horton
Sol Katz
Dick Klingens
Steve Kortendick
Tom Kuiper
Ian Lane
J Willian Leary
Len Lindsay
Dave Lowe

CONTRIBUTORS
David Martin
Edward Matthews
Richard Mayor
Bob McCauley
Rodney McDaniel
Terry Mills
Harald Nendza
Art Paradis
Robert Patry
Kevin Quiggle
Joel Rea
Patrick Roye
Nicholas Seachord
Glynn Stafford
William Staneski
Fred Staudaher
Armando Tamse
Macey Taylor
Colin Thompson
J Thompson
Joe Visser
James White
David Zavitz

From the Editor's Disk

by Len Lindsay

Can we get too much of a good thing? This issue presents a **disk editor** that is one of the best we have seen. Add to that a **3D Projection** system (the airplane is on the cover). Both are for COMAL 0.14.

Then we have a **C128 package** and **Super Chip** for the cartridge owners. Find out first hand just how powerful the Video Display Controller in the C128 really is. Our **VDC'editor** gives you an interactive system for playing with the video display. Set up your screen for 27 lines! Amazing. When you are done, the program prints out all the video register settings so you can use them in your own programs.

Response to our initial announcement of **Super Chip** was fantastic. It may soon be hard to find a cartridge without a chip in its empty socket! Congratulations to the two masterminds behind **Super Chip**: David Stidolph and Richard Bain. It is 16K of Machine Language. That means it is fast. Even Colin Thompson (one of the Beta testers) was surprised at how fast it was!

Now that you have heard about one of his accomplishments, please welcome Richard Bain to *COMAL Today*. Every now and then I take him away from programming to assist in editing the articles. We also wish a fond farewell to Denise. She got a job with a big company. Just what she wanted.

We are continuing to use the small icon pictures that we introduced last issue. They will help you to determine what version of COMAL applies to each article. The **disk** for 0.14, the **cartridge** for 2.0, the **I** for IBM PC COMAL, and the **chip** for 2.0 plus Super Chip.

Thanks to everyone who wrote to AHOY magazine asking for COMAL articles. I think we have their attention. Now it is crucial that the rest of the COMALites write to them (or write again - it can't hurt). Read their letter to us on page 72 of our last issue of *COMAL Today*. Then write to: **Ahoy! / COMAL, 45 W 34 St, Suite 407, New York, NY 10001.**

I hope you kept a copy of your letter to *Ahoy* magazine. Last issue I predicted it would be worth something. Well, it is. Send the copy of your letter to *Ahoy* along with your next order and we will deduct $2. That should help cover your costs. Limit, one per person.

While you are at it, you might send a similar letter to the other two major Commodore magazines: James A Casella, Publisher, Compute!, 825 7th Avenue, New York, NY 10019 and Stephen Twombly, Publisher, RUN, 80 Pine Street, Peterborough, NH 03458

Meanwhile, COMAL is spreading fast. Mark Evans informs us that he has a test version of MacIntosh COMAL from Mytech. The final version is expected in a few months. They are considering converting it to Atari ST next, and then to the Amiga. Mark will try to write an article about it for our next issue.

Paul Ryan, Computer Science, Caulfield Grammar School informs us that COMAL is now the recommended language for schools in Victoria province of Australia. And another disk arrived from the Dutch COMAL Group. They did it again. We haven't had time to try out the programs, but one looks very promising. It "compiles" a procedure into linkable machine code or something of that nature. Hmmmm.

A new project - Geos for COMAL. Ours will be called Ceos. Ideas for it are welcome. The project should begin soon.

COMALites Unite!

by David Stidolph

Last issue I requested some intreped programmer to move the error messages out of the sprite area to the unused memory under the I/O block at $d000 (for COMAL 0.14). Quite frankly, I thought it would be a while before someone could do it. Boy was I wrong!

Robert Ross took up my request and accomplished it in record time. Now error messages take up no sprite space and the code to read the error messages is also hidden in the BASIC routines COMAL uses in the $b000 block of memory.

Also reprinted here is the procedure for expanding COMAL 0.14's workspace to 11,838 bytes free by John H McCoy. These procedures should be added to any "HI" program you may have. They should be called with the following statements:

```
dim ds$ of 2
expand'ram
load'errors
if ds$="00" then change'error'routine
```

Also, please note the **superchip** function listed on page 73. It can be used in your programs to automatically detect if Super Chip is installed or not.

Finally, last issue I mentioned that some people reported problems with my 80 column demo program. The problem turned out to be Commodore. They changed a register within the 80 column chip so that it has to be initialized differently. The C128 package, in this issue and on Super Chip, automatically detects what kind of chip you have and acts accordingly.

```
proc expand'ram
 if peek(18728)<>183 then
  for addr:=45969 to 45981 do
   poke addr+1024,peek(addr)
  endfor addr
  poke 4569,183
  poke 15256,183
  poke 17458,183
  poke 18728,183
  poke 21833,183
  poke 24332,183
  poke 24606,183
  poke 26866,183
  poke 28711,183
  poke 28727,183
  poke 29977,183
  poke 34441,183
  poke 2066,144 // reset top of
  poke 2067,183 // memory pointer
 endif
endproc expand'ram


proc load'errors
 checksum:=0
 for address:=49152 to 49200 do
  read value
  poke address,value
  checksum:+value
 endfor address
 if checksum<>6637 then
  print "load'errors code is incorrect"
  stop
 endif
 open file 8,"comalerrors",read
 ds$:=status$
 if ds$="00" then
  sys 49152
 endif
 close file 8
 data 162,8,32,198,255,169,208,141
 data 32,192,160,0,32,207,255,170
 data 32,183,255,208,24,120,165,1
 data 72,41,240,133,1,138,153,0,208
 data 104,133,1,88,200,208,228,238
 data 32,192,208,223,32,204,255,96
endproc load'errors
//
proc change'error'routine
 checksum:=0
 for loc:=6482 to 6497 do
  read byte
  checksum:+byte
  poke loc,byte
 endfor loc
 for loc:=47006 to 47049 do
  read byte
  checksum:+byte
  poke loc,byte
 endfor loc
 while not eod do
  read address
  poke address+1,158
  poke address+2,183
  checksum:+address
 endwhile
 if checksum<>47040 then
  print "checksum error in data"
  print "* please reload comal *"
  stop
 endif
 // open text "file" for reading
 data 169,0,141,171,183
 data 169,208,141,172,183
 data 169,0,133,144,240,12
 // new routine for "chrin" calls
 data 165,144,208,39,120
 data 165,1,72,41,248
 data 133,1,174,0,208
 data 238,171,183,208,17
 data 238,172,183,169,216
 data 205,172,183,208,7
 data 169,64,133,144,56
 data 176,1,24,104,133
 data 1,138,88,96
 // "chrin" call locations
 data 6510,6521,6529
 data 6538,6543,6548
endproc change'error'routine
```

Submitting Articles

Would you like to share information, programs, or articles with other COMALites? COMAL 0.14 material is especially appreciated. Many COMALites have *moved up* to the cartridge, but there still are more 0.14 users. Send all submissions to:

COMAL Users Group, U.S.A., Limited
6041 Monona Drive
Madison, WI 53716

If you submit a program, please send it on disk. A printed listing of the program is not necessary. If possible, also include a text file explaining the program. Put your name and address as remarks at the beginning of your programs. This helps us give proper credits if they are used. Most important: label the disk with your name, address and date.

Articles should be submitted as standard SEQ text files on disk. If possible, also include a printout of each file on the disk. Don't include any special formatting commands in your files (we have to delete them). We use special formatting with PaperClip for our LaserJet printer.

Don't worry if you aren't a professional writer. Articles sent to us go through extensive editing. We actually go through over 4,000 sheets of paper while preparing one 80 page newsletter! You don't have to follow a bunch of rules, either. We rework your submissions to fit our newsletter format.

Material submitted is not returned, however, if you send us a disk, we will send one of our User Group disks back to you in exchange. Just specify which one.

Submitted material may also be used for our new **READ & RUN** series disks. ■

COMAL On-Line

You can find COMAL support on most of the national on-line networks. Quite a few articles are available to read or download from **Delphi**. COMAL programs are available in the Beyond BASIC section on **CompuServe**. And COMALites meet on **People Link** on Tuesday nights.

After filing for protection from its creditors under Chapter 11 in Bankruptcy Court, **PlayNet** is still operating. While they are operating, we'll continue our national COMAL meeting on the first Thursday of each month, 10pm until midnight Eastern time in the COMAL room.

We also have a national COMAL meeting on **Quantum Link** on the second Thursday of each month in the COMAL room, 10pm to midnight Eastern time. In addition, we maintain an active COMAL section on **QLink** inside CIN under **Magazine Rack**. You can post a question on our Question and Answer board. It should be answered within just a couple days. Meanwhile, you can read all the other Questions as well as their answers! COMAL programs are also available (both 0.14 and 2.0) to download.

There are now several BBS's that have sections for COMAL programs and information. These include:

219-875-6430 - Goshen Town Crier
313-739-1193 - CommoLore (FidoMail 120/5)
313-977-3739 - MCUG BBS
608-784-6500 - Electric Magazine
619-375-6750 - BBS
718-383-8909 - Ahoy BBS (punter protocol)
912-883-7297 - USS Saratoga (24 hours)

We're working on new terminal programs for COMAL 0.14 and 2.0. Meanwhile, Richard Olivieri has released a BBS program written in COMAL 2.0 on a Shareware disk. Anyone interested should try it out. ■

Attention

SCHOOLS

COMAL was designed specifically for use in schools. Elementary schools are pleased with the Logo compatible turtle graphics. High schools are happy with the Pascal structures. Universities find that COMAL prepares students for real world programming.

European schools were quick to recognize COMAL's benefits - both for the teacher as well as the student. Sections of Austrailia now have COMAL as the recommended language. Hundreds of schools in the USA and Canada are switching to COMAL. We provide one free subscription to each school using COMAL (USA only).

Latest news: We soon will have a small book targeted for 2nd and 3rd grade level COMAL programming. Judy Nahman Stouffer has been successfully teaching COMAL to these young children and now has formalized her methods into the book *0.14 Beginners Guide*. It should be available as a book/ disk set soon. Parents should also find it a good book to use with their young children!

Our new spiral bound 265 page 2.0 text book, *Introduction to Computer Programming*, is now in stock! It was written by a teacher in West Virginia specifically for American schools, complete with chapter objectives.

Finally, we have a special, just for teachers using COMAL in their classroom. We have extra boxes, full of backissues 6, 7, 8, and 9 of *COMAL Today* that take up too much space (we need the space for the new books). We will sell these newsletters by the box to schools: $50 per box (about 150 copies). No extra shipping charge if we have a United States UPS address.

USER GROUPS

One of the reasons that COMAL is becoming so popular is the local user groups' support. We wish to continue assisting local groups in helping their members to use their computers effectively. We exchange subscriptions with groups from all over North America. We put the newsletter editor or president of your group on our subscribers list. In exchange, your group sends us a copy of each issue of your newsletter as it is published.

Any material in *COMAL Today* may be reprinted in your local user groups' newsletter. Just give proper credit to the author and *COMAL Today*.

And now we can do even more. If your group is typical, your newsletter editor is vastly overworked. Putting together a newsletter is **hard** work (believe me, I know). So, we have put together a special disk, just for your user groups' newsletter editor. The disk contains SEQ text files of many good articles about COMAL and programming. These files are compatible with Easy Script, Paper Clip (control J), Paper Back Writer, and any other word processor that can accept a standard SEQ text file. The disk is appropriately called Articles Disk. The disk also is an excellent source of information files for a club BBS system.

Could you use the disk? It is available to anyone at the same price as a User Group Disk. But, we will send a free copy of the disk to your user groups' newsletter editor, if he/she will make good use of it in their newsletter or BBS.

In addition, we hope to upload the article text files to QLink, inside the Newsletter Editors Roundtable. ■

Questions & Answers

Ian Lane of Washington, DC has these comments and questions: I have been using the COMAL cartridge on the Commodore 64 for a while now. I am impressed with the speed with which a program can be written to completion-far superior to BASIC, Pascal and C. With the last two, the computer does not help you write the program as readily and there is a lot of waiting for compiling time. "*Try it and see*" is just not as convenient.

I see COMAL very viable for writing programs and then transporting them to other computers. So, on that topic, I have a number of questions:

APPLE COMAL

1. Is COMAL available for the Apple?

Answer: COMAL is not yet available for Apple computers. We hope to have a disk loaded version for the 128K Apple IIe and IIc. We still need another ML programmer.

IBM PC COMAL

2. In what form is IBM COMAL available?

Answer: IBM PC COMAL is available as a disk loaded system.

COMAL COMPILER

3. Is there a compiler available for COMAL for IBM, Apple or Commodore so that a program can be written in COMAL and then compiled to run on these machines without needing a COMAL cartridge/disk? Where can I get these?

Answer: A compiler in not yet available, though we may see one soon for IBM PC COMAL.

IBM COMPATIBLES

4. What about IBM compatibles? Does IBM PC COMAL work for the compatibles such as Kaypro and Leading Edge?

Answer: IBM PC COMAL should work on most compatibles with MS-DOS 2.0 or later and at least 256K memory.

SOURCES

5. Would you please tell me of sources for these requirements? Do you people have programs for compiling COMAL?

Answer: IBM PC COMAL is available from our group. We import it from IBM Denmark. COMAL Today #11 listed all the COMAL implementors.

Doug Colpitts of John Norquay Elementary School has these questions:

EPSON DUMP

1. May we have an MX-80 graphic screen dump in *COMAL Today*?

Answer: See COMAL Today #10, page 66 and 72 for two COMAL 2.0 packages for Epson. Epson dumps are also on Utilities Disk #1 and Utilities Set #2.

WINDOW

2. What are the WINDOW settings to make a CIRCLE be a circle and not an ellipse?

Answer: This varies, depending on your monitor. Printers also vary in the settings needed. Use the program on page 79 of COMAL Today #12 to test specific settings for your monitor.

More ►

CIRCLE

1) What is an easy way to draw a circle on the graphics screen?

Answer: COMAL 2.0 includes a CIRCLE command. From COMAL 0.14 it could be:

```
setgraphic 0
length=3; angle=5
for x=1 to 360 step angle do
  forward length
  left angle
endfor x
```

This will look like an oval since the screen is not at a 1 to 1 ratio. Accurate circles can be drawn using routines in the book Library of Functions and Procedures.

DIRECTORY FROM A PROGRAM

2) How can I show or load the directory from a program?

Answer: In COMAL 2.0, the DIR command will display the directory. COMAL Today #10, page 28, has a routine for COMAL 0.14.

DISK FORMAT

3) How do I format a disk within a program using a variable for the disk name and id?

Answer: the following lines will do it:

```
dim name$ of 16, id$ of 2
input "Disk name: ":name$
id$=chr$(rnd(65,90))+chr$(rnd(65,90))
pass "n0:"+name$+","+id$ //for drive 0
```

SCRATCH FILES

4) How do I scratch a file from within a program using a variable?

Answer: the DELETE command will do it:

```
dim filename$ of 16
filename$="oldfile"
delete "0:"+filename$ // for drive 0
```

SIZZLE

5) Can sizzle work with the Enhancer 2000 disk drive?

Answer: sizzle is only for the 1541 & 1571.

PARADISE <> SUBSCRIBER

6) Since the Programmers Paradise comes with 6 *COMAL Today* newsletters, does that make me a subscriber? Do I get discount prices? - Dave Egts, Johnstown, PA

Answer: No. To be a subscriber you must actually subscribe. We think it is worth subscribing for the information - but after you buy a few items, the discounts may actually exceed the original subscription cost.

ACCURACY

Question: I was exploring the PRINT USING statement and found that anything over about 9 digits long produces an error:

```
10 input num
20 print using "##########.###": num
```

When run, an input of 123456789.000 yields 123456788.992. Is it due to some kind of roundoff error? - Nicholas Seachord, Seattle, WA

Answer: Yes. The way C64 COMAL (and C64 BASIC) store real numbers results in roundoff errors. The accuracy level is about 8-9 digits.

More ►

ORIGINAL BEIGE 2.0 CARTS

Question: Since my beige COMAL 2.0 cartridge won't work in my C128, I have been reduced to writing in C, which isn't as much fun. Any word on a better power supply for the C128? Anyone interested in swapping a black cartridge for a grey one? Or should I consider buying a new cartridge? - Art Paradis, Anaheim, CA

Answer: Both beige & black 2.0 cartridges are version 2.01. The black cartridge uses two 32K ROMs while the beige cartridge uses four 16K EPROMs (which require twice as much power). The black cartridge also includes an empty socket.

We have plans that detail how to modify the beige cartridge so that it will accept two 32K EPROMs, leaving two empty sockets! This modification (NOT for novices) makes it more flexible than the black cartridge, since two custom EPROMs can be added.

There now are several options for those with grey cartridges who need either the empty socket or a solution to the cartridge power requirements.

1) Have your cartridge modified to use two 32K EPROMs.

2) Get a new power supply that has more power for the computer system.

3) Get a new black COMAL 2.0 cartridge. We now sell the cartridge plain (no manuals or disks). Your old cartridge can be donated to your local school (probably tax deductible). Or sell it for $35, or so, to help make up for the new cartridge price. If you can't find anyone in your area to buy it, we will print your name, address, phone number and asking price in COMAL Today.

EASY READING

Question: Your laser printer is lovely. But how come you don't right justify? - Macey Taylor, Tucson, AZ

Answer: We are trying to keep COMAL Today as readable as possible. Studies have shown that it is easier to read text that is ragged right rather than justified. This has to do with how you read. The ragged edge gives your eyes "landmarks" to keep in line.

Research also found that it was easier to read text in two columns per page rather than one. This has to do with the distance from the end of one line to the start of the continuation line. If your eyes must "jump" too far, it becomes harder to read.

Also, you may have noticed that we now are using a Times Roman typestyle for our articles. It is supposed to be one of the easiest to read. Plus, our non-glossy paper is easier on your eyes since there is no "glare".

We also include a blank line after each paragraph. This extra white space provides a more pleasing look to a page, increasing its readability.

Finally, I hope you noticed the lack of "continued on page 44" type breaks in articles. Studies have shown that breaking up articles into two or more sections is one of the most frustrating things. We finish one article before beginning the next.

We are striving to not only provide you with useful COMAL information - but to provide it in the most readable form we can. Please let us know if you have suggestions for improvements.

More ►

DEVICE 9 PASS

Question: I have two 1541 drives. How can I use the PASS command with device 9?

Answer: PASS will work only with device 8 in COMAL 0.14. From COMAL 2.0 device 8 is the default and you can specify another device like this:

```
pass "n0:disk,id",9
current'device:=10
command$:="n0:workdisk"
pass command$,current'device
```

TIME

Question: Is there any way to access BASIC's **TI** (real time clock) variable? - Reed Brown, Collinsville, CT

Answer: COMAL 2.0 has a command **TIME** *that is equivalent to the BASIC command* **TI.** *COMAL 0.14 does not have the command built in. You can add it using this function:*

```
func ti closed
  j:=256*256*peek(160)+256*peek(161)+peek(162)
  return j
endfunc ti
```

Now, to use it:

```
print ti
```

COMAL IN SCHOOLS

Question: Is there a listing of elementary, middle, high schools or colleges which use COMAL in course work? - Rodney McDaniel, Jonesboro, AR

Answer: We hope to create such a list soon. There are over 200 schools in Canada and the USA using COMAL already.

Armando Tamse has these questions:

8032 COMAL

Question: Is there a COMAL version available for the CBM 8032?

Answer: Yes. PET COMAL 0.14 is available in 4040 disk format for $14.95. Add $10 service charge to have it copied to 8050/8250 disk format.

GRAPHICS WITH COMAL

Question: Other than the C64 versions, what other COMAL versions for CBM computers have a graphics capability? How about for other computers?

Answer: There is a COMAL 2.0 plug in ROM board for the CBM 8032 and 4032 computers. There is an optional hi-res graphics board that can plug into it. A special graphics package is available for use with this set up. The boards are of extremely high quality. The price is not cheap, possibly between $300 and $500 each. Other computers? IBM PC COMAL from IBM Denmark (UniComal) and Mytech Data both include a high resolution graphics system. Many of the COMAL implementations for the European computers include graphics controls.

PRINT USING

Question: Is it possible to output numbers with the PRINT USING statement with commas for every third number such as 34,000,000.00?

Answer: While this option is not available now, the possiblity is being studied by the COMAL Standardization Group (members are from companies with COMAL implementations).

More ►

2.0 FUNCTION KEYS

Question: How can I use DEFKEY or another method to print drawto(on the next program line after hitting the <return> key on the previous line? Can a single keystroke, such as **F3** cause drawto(to be entered so that I don't have to type it over and over? - Robert Gerber, Racine, WI

Answer: We can set up the F3 key to not only print the drawto(for you, but the) and <return> on the previous line as well. Just type this:

```
USE system
defkey(3,")"+chr$(13)+"drawto(")
```

That's it. Now, when typing in a series of drawto statements, you only need to type the numbers! For example, type this:

```
auto 5000
5000 //          don't hit return
```

Don't hit the return key - press **F3**. *This is what your screen looks like now:*

```
5000 //)
5010 drawto(_
```

The cursor is waiting for you to type in the two numbers. Type in the numbers - and hit **F3** *after the second number:*

```
5010 drawto(44,71
```

After you hit **F3** *your screen looks like:*

```
5000 //)
5010 drawto(44,71)
5020 drawto(_
```

Now, just keep typing in two numbers, followed by the **F3** *key, until you are done entering your drawto statements. The reason we started with a // for line 5000 was to remark out the first end parenthesis. You can delete that line now if you wish.*

Further Reference:

2.0 Auto Save, COMAL Today #11, page 55
Function Keys for MSD Dual, COMAL Today #9, page 32
Function Keys (three notes), COMAL Today #9, page 33
Function Keys, COMAL Today #8, page 33 and fix in COMAL Today #9, page 7
Redefine Function Keys, COMAL Today #7, page 21

CARTRIDGE BOOKS?

Question: Can you recommend other books to augment the *Cartridge Tutorial Binder*? I find the binder lacking in its treatment of machine language. Other books might explain in more depth a wide variety of subjects. - William Staneski, Suffolk, VA

Answer: There are several other books that you might find helpful:

COMAL 2.0 Packages by Jesse Knight - explains how to create your own machine code packages for use with COMAL 2.0 programs.

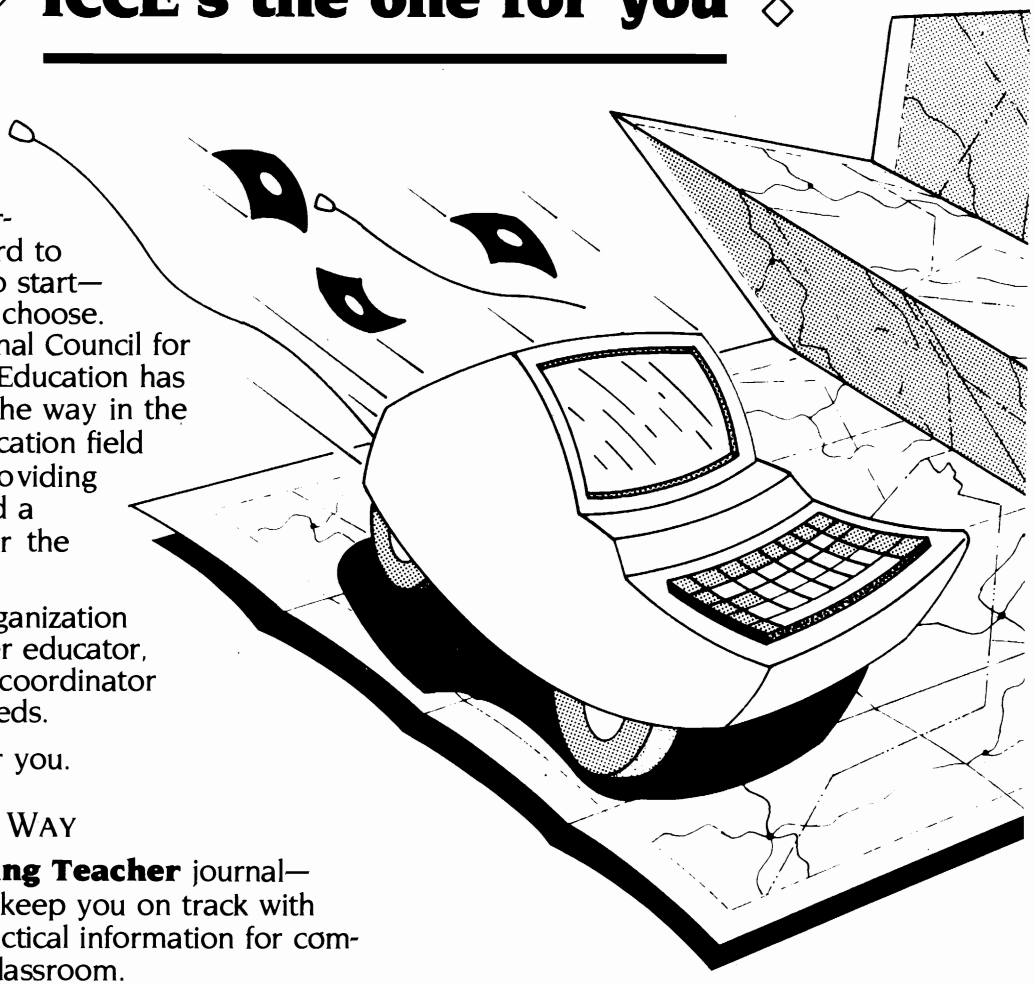
Packages Library by David Stidolph - presents 17 different packages, ready to LINK and USE, many with source code.

Introduction to Computer Programming with C64 COMAL 2.0 by J. William Leary - textbook introduction to programming using COMAL 2.0.

COMAL Handbook by Len Lindsay - the detailed reference to COMAL 2.0. ■

Take Off With Us

◇ ICCE's the one for you ◇



With today's profusion of computer information, it's hard to know where to start—which road to choose. The International Council for Computers in Education has been guiding the way in the computer education field since 1979, providing leadership and a ground plan for the future.

It's the one organization every computer educator, administrator, coordinator or librarian needs.

It's the one for you.

Guiding the Way

The Computing Teacher journal—guaranteed to keep you on track with up-to-date, practical information for computers in the classroom.

Special Interest Groups—share information to help your special interest area grow. SIGs include computer coordinators, teacher educators, administrators and special educators, and are planned for advanced placement in computer science, community colleges and videodisc users. The quarterly **SIG Bulletin** serves as a forum for SIG information.

Booklets and Monographs point to additional information on specific topics. **ICCE Packets** provide you with teacher training materials. Members receive a 10% discount on all three.

ICCE Committees address a variety of ethical and practical issues important to you as a computer-using educator.

ICCE participates in computer education conferences throughout the world, supporting our state- and region-wide member organizations.

International Council for Computers in Education — ICCE

Join the One for You.

Letters

GOOD AND BAD NEWS

Dear Mr. Lindsay- I have some good news and some bad news. The good news is that my COMAL 2.0 cartridge works impressively. My bad news is that I have not yet figured out a way to make the COMAL cartridge and my Skyles Quicksilver interface cooperate in any fashion. The interface apparently takes control of the C64 on reset, and puts a new KERNAL in place. It appears that this revised KERNAL does not check for the presence of a cartridge (COMAL 2.0, anyway), so the computer comes up with the special Quicksilver startup message, rather than in COMAL.

My own salvation in this situation is based on two factors: 1) there is a switch to turn the interface off, so the original KERNAL is in place, and 2) I came upon a used and abused 1541 for $50 a couple of weeks ago, repaired it, and already had it in place, running some software that also would not run with Quicksilver. So I am out of the woods, but other Quicksilver owners who want to use COMAL 2.0 ought to be warned of the situation. (Also, the 2.0 users who are contemplating purchasing Quicksilver).

Quicksilver and COMAL 0.14 get along famously, except that

CHAIN "name",9

Gives a "not implemented" error, and

CAT 0,9
PASS "(anything)",9

Both give syntax errors, with the cursor landing on the comma. So I turn off one drive and use the other one as #8. Not perfect, but it works. *[Yes, COMAL 0.14 cannot do these things, even with a 1541].*

You may be interested in a planned application of COMAL 2.0. The University is purchasing a 2.0 Deluxe Cartridge Pak for me to have at school, so I can determine the desirability of controlling operation of a milling machine I have interfaced to a C64 in the COMAL language.

The BASIC program used at present works well enough, except for the high-resolution graphics verification of the tool centerline path which I borrowed from Compute!'s Gazette, with permission. When I saw the commands DRAWTO and MOVETO in COMAL, it all but took my breath away; the BASIC algorithm for plotting the tool path was becoming increasingly complex as I have attempted to keep ahead of increasingly enthusiastic student efforts. Since I am planning for success in my COMAL efforts, you may plan on sending more cartridges and a selection of books and software this way within the next few months. - Edward Matthews, Springfield

ANOTHER COMALite

Dear Sirs- Greetings and Salutations from the **MANIC MECHANIC.**

After a concerted study of Roy Atherton's book *Structured Programming with COMAL*, as well as varied perusal through pertinent articles, it has become readily apparent to me that COMAL is to BASIC what Homosapiens are to Neanderthal Man. It would be the understatement of the century to say that I am anxious to abdicate any further studies in BASIC in favor of COMAL. - J. C. Thompson, Clarckston, GA

More ►

MACHINE LANGUAGE

I'd like to see an expanded treatment of machine language and do-it-yourself ML packages in *COMAL Today*. My mind is getting old and soft, and the treatment in the manual wasn't crystal clear. I do have some ML routines I wish to be able to load easily and just SYS from within the COMAL program. Regards, and I Love COMAL! - Bob McCauley, APO, NY

I hope you like the bits of Machine Language we have been adding in COMAL Today. And SYS is not needed to access your ML in COMAL 2.0 if you make them into packages. With packages you can call your ML routines by name!

COMAL INTEREST

Dear Editor: The interest COMAL gets at a meeting and on the BBSs (I'm also running the COMAL section on the club's BBS) is astonishing. Sometimes I wonder if it is a matter of power perception. Computers seem to be in charge of big parts of our lives: bills, paychecks, etc. When people see the power they can have over the behavior of a computer through COMAL programming, how simple it can be, and that it isn't mysterious or magic, they get excited.

It seems like such a waste to see people possessing a piece of equipment as powerful as a C64, a unit we could not have had at any price ten or fifteen years ago when I learned BASIC on a mainframe, and using it for playing Donkey Kong and Chilly Willy. To have them interested in any kind of programming seems much healthier. (Sounds like I've taken the Protestant Ethic to heart.) I guess that's why I'm so enthusiastic about COMAL. - Ed Matthews, Springfield, MO

COMAL GREAT IN SCHOOL

Dear Editor: We use 32 C64s in our computer lab. The high schools in Evansville are programming in Pascal, but there is no way that middle school students can handle that! Besides that, there is the BASIC block to contend with as they try to **unlearn** bad habits they have picked up with BASIC. COMAL seems to be the ideal language to use in the middle school ... maybe also in the high school, but that's another universe. COMAL's structure and modular design is more like the other strandard programs used in today's work world. In addition, it would be helpful, even if the high schools continue to use Pascal. - Robert P Patry, Plaza Park Middle School, Evansville, IN 47715

INTEREST IN PROGRAMMING

Dear Editor: I'm a computer language nut and upgraded from the C64 to the C128 to be able to play with the languages under CP/M (Turbo Pascal, PL/I-80, FTL, Modula-2, etc). I like what you are doing with COMAL. Many people are frustrated with BASIC. COMAL seems to be bringing back an interest in programming. Your magazine, *COMAL Today*, shows COMALites experimenting with more features of the Commodore computers than the BASIC programmers. I would like to play with the package feature of COMAL 2.0 (I still have COMAL 0.14). - Glynn Stafford, Waldorf, MD

LIKE A NEW MACHINE

Friends- I have never been more pleased with my C64. It's like a new machine. Unfortunately I can't bear to do anything in BASIC anymore. Ah, Well! Love that Cartridge!! Many Thanks. - Dan Horton, Northampton, MA ■

COMAL Clinic

INKEY 0.14

Mike Erskine has provided us with the following procedure for COMAL 0.14 that will wait and get the next keystroke:

```
proc inkey(ref c$) closed
  while key$>chr$(0) do null//clear buffer
  repeat
    c$=key$
  until c$>chr$(0)
endproc inkey
```

An example of using it is:

```
inkey(reply$)
```

2.0 FEATURES DISCOVERED

I just discovered another power-packed set of features in the cartridge. On the next to last page of *Cartridge Graphics and Sound*, the additional functions with the CONTROL key are great. Textscreen dump just by entering CTRL-P is fabulous. And you can get this dump in the middle of a program. In fact, I got a screen dump in the Plot'Function program while the program was waiting for an input.

I had previously discovered that when editing a program with the automatic COMAL indentions that an entire PRINT statement had to be re-entered if it occupied more than one line, or the indentations would also be printed. I was delighted to discover that before editing an indented PRINT statement all I had to do was enter CTRL-A and the indentions were removed.

You cannot emphasize enough the importance of getting the *Cartridge Graphics and Sound* book with the cartridge. It contains valuable information. - Dave Lowe, Long Beach, CA

FILE EXISTS

It is a good practice for a program to check if a file is actually on the disk that is in the drive before trying to read from it. Reading from a non-existant file is meaningless. You can have your program check if a file exists, or not, with the function shown below. A different version is needed for 0.14 and 2.0 due to the way disk errors are handled.

COMAL 0.14

```
func file'exists(file'name$) closed
  dim s$ of 2
  open file 78,file'name$,read
  s$:=status$
  close file 78
  if s$="00" then
    return true
  else
    return false
  endif
endfunc file'exists
```

COMAL 2.0

```
FUNC file'exists(file'name$) CLOSED
  TRAP
    OPEN FILE 78,file'name$,READ
    CLOSE FILE 78
    RETURN TRUE
  HANDLER
    CLOSE FILE 78
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
```

CENTER 2.0

Here is a quick way to center text:

```
proc center(text$) closed
  print at 0,(40-len(text$))/2: text$
endproc center
```

More ►

STRING LENGTH NOTES

A DIM statement is used to allocate space for use by a string variable. However, the length of the string assigned to the variable can vary from 0 (null string) to the length specified in the DIM statement. In COMAL 0.14 a DIM is required for each string variable used. COMAL 2.0 makes this optional - if you don't do the DIM, COMAL 2.0 will do it for you, for a maximum length of 40. Here is an example to demonstrate the variable length:

```
dim text$ of 20
input "Enter short word: ": text$
print ">",text$,"< is";len(text$);"long"
```

RUN

Enter short word: testing
>testing< is 7 long

If you want **text$** to always be 20 characters long, you can do it like this (text$ must be DIMed before use like this):

```
dim text$ of 20
input "Enter short word: ": text$(1:20)
print ">",text$,"< is";len(text$);"long"
```

RUN

Enter short word: testing
>testing < is 20 long

Notice that spaces are automatically padded at the end of the string you entered to fill the space requested by the (1:20) we tacked onto text$. This is a substring specification. When assigning a substring, COMAL will always pad spaces to the right if more characters are needed than provided.

0.14 DYNAMIC NEW

Terry Mills of Hanover Park, IL provides a short routine that allows programs to erase themselves when finished running. Just put this at the end of the program:

```
print chr$(147),"new"
print chr$(17),"size"
print
poke 631,19
poke 632,13
poke 633,13
poke 634,13
poke 635,17
poke 198,5
end
```

We took his idea and modified it to just do a NEW followed by a clear screen:

```
print chr$(147),"new"
poke 631,19
poke 632,13
poke 633,147
poke 198,3
end
```

CALL THE NEXT PROGRAM

In COMAL 0.14, if there is a large program in memory when you CHAIN another large program, there may be a memory problem. This CALL procedure uses the method suggested above so that a NEW precedes the CHAIN. This will work in all cases.

```
proc call(filename$) closed
  print chr$(147),"new"
  print chr$(17),"chain""",filename$,""""
  poke 631,19
  for i:=632 to 634 do poke i,13
  poke 198,4
  end
endproc call
```

COMAL Structures While Loops

by Richard Bain

Programmers who are new to COMAL will find many new programming structures to make programs easier to write and understand. The **WHILE** loop is one of these. Here are some hints on when to use the **WHILE** loop, how to use it, and what similar structures may be used instead.

The **WHILE** loop should be used in cases when the programmer or user would think in English, *"While this is happening, I should be doing something."* Let's see how this might look in a few real programming situations:

```
while not eod do
  read number
  print number
endwhile
```

This program fragment will read numbers from data statements and print them out. It automatically exits the loop when there is no more data to read. It could easily be changed to do almost anything else with the numbers, but the important point to remember is that the amount of numbers does not matter. Data statements can be added or deleted, but the **WHILE** loop doesn't need to be changed.

```
while raining do play'inside
```

This is an example of a one line **WHILE** loop. The differences between this loop and the loop above are that you can only use one statement after the **DO**, and you don't type in **ENDWHILE**. In this example, **raining** can either be a simple variable or a function call which **returns** a 1 for **TRUE** or a 0 for **FALSE**. **Play'inside** is a **procedure** call. This may be a menu to let you choose one of many computer games, or it could just be the name of your favorite game. That's up to you. Even though **play'inside** is only one statement, being a **procedure** call allows it to do many things. When it stops raining, you will no longer be nagged to stay inside. See, it's almost like English.

```
while key$ = chr$(0) do null
```

This is a useful line which you may already have seen in several programs. It is simply a delay loop. Usually, a message will first be printed on the screen to tell the user to hit a key to continue. Then this loop makes the program pause until a key is pressed. This is useful in many cases such as a text reading program in which the user will want to pause before unread text scrolls off the screen. Did you know that doing nothing (**NULL**) could be so useful?

Now, let's compare the above examples to similar examples with other structures. The first one could be replaced with a **FOR** loop in many cases. You would want to do this if you knew exactly how many numbers to read in, or if you wanted to store the data in an *array* and needed a counter for the *array* index.

In the second example, many people might be tempted to use a **REPEAT** loop. This is not a good idea. What if it isn't raining at all. The **REPEAT** loop will always make you play inside for a little since a **REPEAT** loop *always* executes at least once. The **WHILE** loop will be skipped over completely if it isn't raining.

BASIC only has a **FOR** loop. Some experienced BASIC programmers will simulate **WHILE** and **REPEAT** loops with those awful **GOTO** statements, but in COMAL you don't need to. Now, aren't you glad you switched to COMAL. ■

Now Available Through Aquarian Software

Gold Disk Series

Volumes 1 through 11 Now Available!!!

Volume 11 Features a C-64 Assembler

Each Disk Contains:

- The Monthly Feature Program
- Programming Tutorials
- High Quality Games
- And Much More

Gold Disk Series for 128
Coming Soon!

Only $14.95 Per Disk*

* Plus Shipping and Handling

The Cataloger

The Ultimate Disk Cataloging System for the 64!

Features of The Cataloger V3.5A include:

- ★ Loads directly from the disk itself.
- ★ Ability to change name of entry.
- ★ Fast — Uses relative files exclusively
- ★ Search, Sort and Print by any of 12 fields.
- ★ 1100-program (or disk) capacity per data disk.
- ★ All machine language.
- ★ Menu driven — very easy to use.
- ★ Works with one or two drives.

Only $24.95

BobsTerm Pro

The Ultimate Terminal Software !

Upload / Download Supports Punter, X-Modem, XON / XOFF, DC1 / DC2, and Much More!

28.5 Byte Buffer with unmatched editing abilities

- User Adjustable Parameters
- 10 Custom Character Sets
- Unlimitied Phone Book Storage
- Programmable Macro Command Strings

Only $59.95

MATRIX — NOW AVAILABLE!!

The Indispensable C-128 Utility / Starter Kit !

Use **dozens** of 128 features in the 64 mode:

- Numeric Key Pad
- Cursor Keys
- 80-Column RGB Output
- Many Other Special Function keys

One-Key Functions include:

- **2 Megahertz "Fast Mode"**
- **One-Key Screen Dumps**
- **Full-Featured DOS Utility Menu**

Other Features include:

- Fast Loading
- Fast Copy For The 1571!
- Relocatable In Memory
- 100% Transparent to BASIC

Available Now For Only $59.95

Graphic Screen Exporter

A Universal Graphics Converter !

Converts Anything to Anything — Including:

Koala Pad	Doodle
Flexidraw	Print Shop
COMAL	Paint Magic
CAD GEM	Micron Eye

And Many Many More ! !

The Most Versatile Graphics Utility Ever Released for the Commodore 64 !

Only $29.95

ALSO AVAILABLE:

OmiTerm.....$19.95
Full-Feature Terminal at an Affordable Price!

Turbo Calc/64.....$17.95
A great spreadsheet at an Unbelievable Price!

Tax Computation.....$29.95
The friendliest tax package on the market.

Guitar Master.....$49.95
A comprehensive musical instruction package

Fast Boot!.....$14.95
Mike J. Henry's Fast Loader for 1541/MSD

Thriller Collection.....$24.95
Seven intricate text adventures on one disk

Call or Write for Full Catalog !

CAD-GEM

Computer Assisted Design
Graphic Element Manipulation

A Wire Frame CAD system for the C64 !
Input from Joystick, Track Ball, Light Pen or Graphics Tablet
360 Degree Rotation in .1 Degree Increments
Scaling on a 64K x 64K, 2048 Mega-Bit Virtual Screen
Independent Manipulation of 400 Objects (Points or Lines)

You must see CAD GEM to believe it!
Demo Disk Available for $3.00

$89.95

MODEM MASTER

The Friendliest Commodore BBS Available

- Works with 1541 or MSD Dual Drive
- 300 / 1200 Baud Operation
- New Punter File Transfer Protocol
- Sub-Directories for File Transfer
- 250 User Capacity
- Accurate Clock / Calendar
- Printer Output
- Information Files
- "Old" E-Mail Deleted After One Week
- Set Up In Only 10 Minutes !

Only $29.95

Total Software Development System

by Kevin Pickell
Now Available in the States !

Assembler/Editor — fast load, get, log and loadat; adds 38 new commands; full macro instructions; allows 13-character labels; assembles to and from disk
Sprite Editor — 256 sprites in memory, view 64 at same time, works with keyboard, joystick or trackball, animates sprites during design
Unassembler — create source code from any ML program
Sound Editor — create interrupt-driven sound effects
Character Editor — edit all characters. Screens to 255x64. Hi-res & Multi-color Character Sets

TSDS automatically includes sprites, characters, mattes and sound effects into source code!

Only $39.95

128 Version Coming Soon !

Aquarian Software
P.O. Box 22184
Portland, OR 97222

VISA

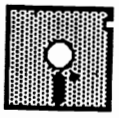
To order, Call: (503) 654-2641
VISA & MasterCard Accepted

MasterCard

Add 3.00 S & H Per Order
(Add Additional $2.00 for COD)
Canadian Orders Add 10.00 S&H
Allow 3-4 Weeks For Delivery

Write or Call for Full Catalog — Dealer Inquiries Welcome !

Quick Sprites



by Sol Katz

Just about everyone who owns a C64 wants to create and move sprites. Unlike BASIC, COMAL was designed to help you do just that. Leaving out all the computer jargon, a sprite is a drawing made on a piece of graph paper that is 24 squares across and 21 squares down. Each square is either colored or not. (I'll leave multicolored sprites for a future column). Since computers can handle data conversion easily, I wrote a program that "reads" a drawing of a sprite that was made from data statements. Each statement must be a 24 character string. With a little imagination, it should look like an animal.

```
data "                      OO      "
data "                     OOOO     "
data "                    OOOOOO    "
data "                   OOOOOO     "
data "                  OOO         "
data "                  OOO         "
data "                  OOO         "
data "                 OOO          "
data "    OO           OOO          "
data "   O  O          OOO          "
data "O                 OOO          "
data "O     OOOOOOOOOOOOOO          "
data " OOOOOOOOOOOOOOOOOO           "
data "  OOOOOOOOOOOOOOOOOO          "
data "   OOOOOOOOOOOOOOOOOO         "
data "   OOOOOOOOOOOOOOOOOO         "
data "   OOO OOO     OOOOOO         "
data "   OO  OO       OO OO         "
data "   O  O          O   O        "
data "  O  O          O      O      "
data " OO   OO       OO       OO    "
```

The following is computer jargon. You won't need it to make sprites, but it explains what is happening. We need to convert the data statements into an image definition. The computer stores an image definition in 64 bytes, each byte having 8 bits. 63 bytes are used for the image definition, the last byte determines if the image will be hi-res or multicolor. Looking at the data statements, each statement has 24 characters (either non-space or space). Each character represents 1 bit, so that the 24 characters equal 3 bytes. We read the 24 characters into a 24 character string variable called **row$**. The program will need a **DIM** statement for **row$** since COMAL 0.14 requires all strings to be dimensioned.

We will have to read 21 data statements, so we will use a FOR..ENDFOR loop with 21 repetitions (iterations). Within this loop, we need a second loop to read the data statement's 24 characters. It converts them into 3 characters (24 bits) for the sprite definition. We put in a third interior FOR...ENDFOR loop which will keep doubling the place value (I used addition, which is faster than multiplication or exponentiation), as we move from the least significant (right most) to the most significant (left most) bit. If the character is not a space, we add the place value to the total. If it is a space, nothing is added to the total. This is, in fact, binary arithmetic, with spaces as zeros and non-spaces as ones. After 8 bits are totaled, the decimal value is converted to a character equivalent and added to the end of the 64 character string (called **definition$**) that will become an image definition (**definition$** will also have to be dimensioned). All the foregoing rigamarole is to allow you to use the image you drew with the 24 x 21 grid of data statements. We then add the last byte, using 0 for a hi-res image.

With much more difficulty you could have drawn on a paper grid and calculated the numeric value of each byte and put those 64 values into data statements (which is a more traditional approach) or you could have used a special sprite generation program. The

More ►

program presented here uses 16 lines of code to convert our data statements into image definitions.

Now back to English...

The following code will convert the 21 data statements into an image:

```
dim row$ of 24, definition$ of 64
definition$:=""
for line:=1 to 21 do
  read row$
  hi:=8; low:=1
  for byte:=1 to 3 do
    decimal:=0; value:=1
    for place:=hi to low step -1 do
      if row$(place)<>" "then decimal:+value
      value:+value
    endfor place
    hi:+8; low:+8
    definition$:=definition$+chr$(decimal)
  endfor byte
endfor line
definition$:=definition$+chr$(0)
```

Once the image defintion is created, we need to use the COMAL sprite commands to first make the sprite and then make the sprite move.

First, we enter hi-res graphics mode with:

```
setgraphic 0
```

Then, tell the computer that image definition #1 looks like the shape we just built and called **definition$**:

```
define 1, definition$
```

Next, tell the computer that sprite 0 will have the shape that was defined as 1:

```
identify 0,1
```

Within a running program, you can change the identity of any sprite so that the sprite may use different image definitions.

We set the color of the sprite with:

```
spritecolor 0,8
```

Now sprite 0 is orange. We set the size of the sprite with:

```
spritesize 0,1,1
```

(sprite 0, double wide, double high) and sprite position with:

```
spritepos 0,50,100
```

(sprite 0, 50 across, 100 up). We will hide the turtle using the **hideturtle** command. Here are all the lines:

```
setgraphic 0
hideturtle
define 1,definition$
identify 0,1
spritecolor 0,8
spritesize 0,1,1
spritepos 0,50,100
```

At this point, we have a sprite on the screen. We can toggle back and forth between the text screen and graphics screen using F1 and F5. The only thing left to do is move the sprite. Since **spritepos #,x,y** controls the sprite position on the screen, we'll use a loop to change the x and y values. We'll move the sprite diagonally across the screen, showing the 1.5 to 1 relationship of x to y pixels (about 320 to 210). We now add the necessary code to move the sprite up one and over two for each iteration of the loop.

```
for move:=1 to 200 do
  spritepos 0, move+move/2,move
endfor move
```

More ►

For this special case of only one line in a loop, COMAL allows a short cut.

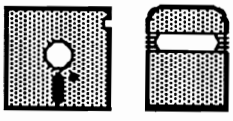
```
for move:=1 to 200 do spritepos 0,move+move/2,move
```

Other sprite commands available in COMAL include: datacollision, spritecollision, priority and hidesprite.

Further Reference:

COMAL 2.0 Sprite Quirks, COMAL Today #10, page 24
Ready Aim Draw, COMAL Today #10, page 43
Walker- Sprites on the March, COMAL Today #10, page 44
Quick Change Sprite, COMAL Today #9, page 30
Tanks & the Animate Keyword, COMAL Today #9, page 64
Read Sprite, COMAL Today #5 page 26
Sprites and COMAL, COMAL Today #1, page 15
Sprite Statements Chart, COMAL Today #1, page 16 ■

Zoo Match Game



by Ed Bolton

Zoo'match'game is a fun game for one person that lets you try to find matching pairs. You can choose to match pictures to pictures, or pictures to words. To choose a picture, follow the menus and type the letter for the square you want. When you find a pair, you will be in for a pleasant surprise (the animals are animated when you correctly match a pair). After the game ends, you have a choice to quit or to play again. The game is on *Today Disk #13* for both versions of COMAL. ■

Wheel of Fortune

```
// save  "wheel'of'fortune"
//  by bob hoerter
DIM name$ OF 30, letter$ OF 1
DIM used$ OF 200, t$ OF 2, wheel'$ OF 92
PAGE
PRINT CHR$(14),CHR$(8),
USE system
select'word
get'players
play'game
show'word
textcolors(-1,-1,8)
CURSOR 22,8
PRINT "Oh groovy!!"9"";player$(zz);
END "you won $"+STR$(total'win(zz))+"."
//
PROC select'word
 RANDOMIZE
 textcolors(0,0,3)
 zip'text(4,"WELCOME")
 zip'text(10,"to the")
 zip'text(16,"WHEEL of FORTUNE")
 FOR temp:=1 TO 1000 DO NULL
 textcolors(-1,-1,15)
 subject:=RND(1,3)
 CASE subject OF
 WHEN 1
  clue$:="PERSON"
  RESTORE people
  FOR numb:=1 TO RND(1,20) DO READ name$
 WHEN 2
  clue$:="PLACE"
  RESTORE place
  FOR numb:=1 TO RND(1,20) DO READ name$
 WHEN 3
  clue$:="THING"
  RESTORE thing
  FOR numb:=1 TO RND(1,20) DO READ name$
 ENDCASE
 length:=LEN(name$); r:=6; c:=30
 DIM answer$ OF length, remember$ OF length
 prompt
 remember$:=name$
 used$:=""
 PAGE
ENDPROC select'word
//
PROC new'letter
 PRINT AT 3,8: SPC$(22)
 show'word
 found:=0; letter$:=""; new'win(zz):=0
 format
 PRINT AT 19,1: "Choose Letter:",SPC$(65),
 CURSOR 19,16
 REPEAT
  letter$:=inkey$
  chk'for'vowel
 UNTIL letter$>""
 PRINT AT 19,1: SPC$(79)
 FOR p:=1 TO length DO
  position:=letter$ IN name$
  IF position>0 AND position<=length THEN
   answer$(position):=letter$
```

More ►

```
    name$(position):="#"
    found:+1
   ELSE
    total'win(zz):=0
   ENDIF
   IF found THEN
    right:=TRUE
   ELSE
    right:=FALSE
   ENDIF
  ENDFOR p
  print'guess
ENDPROC new'letter
//
PROC show'word
  textcolors(-1,-1,15)
  PRINT AT 2,1: "CLUE >>";answer$
  PRINT AT 3,0: clue$
ENDPROC show'word
//
PROC prompt
  FOR l:=1 TO length DO
   CASE name$(l) OF
   WHEN "'"
    answer$(l):="'"
   WHEN " "
    answer$(l):=" "
   OTHERWISE
    answer$(l):="-"
   ENDCASE
  ENDFOR l
ENDPROC prompt
//
PROC get'players
  PAGE
  REPEAT
   TRAP
    INPUT AT 2,5,1: "Number of players (max 4)> ": np
   HANDLER
    PRINT AT 6,5: "Numbers only!"
    PRINT AT 2,32: SPC$(2)
   ENDTRAP
  UNTIL np>0 AND np<5
  PRINT
  DIM player$(np) OF 9, win(np), new'win(np), total'win(np)
  DIM player'color(np)
  FOR n:=1 TO np DO
   INPUT AT n+5,5,9: "Name please > ": player$(n)
  ENDFOR n
  PAGE
ENDPROC get'players
//
PROC play'game
  LOOP
   FOR turn:=1 TO np DO
    zz:=turn
    format
    PRINT AT row,col: player$(zz)
    REPEAT
     show'word
     make'choice
     new'letter
     winnings
     EXIT WHEN answer$=remember$
    UNTIL right=FALSE
   ENDFOR turn
  ENDLOOP
ENDPROC play'game
//
PROC format
  row:=14
  CASE zz OF
  WHEN 1
   player'color(zz):=7
   col:=1
  WHEN 2
   player'color(zz):=2
   col:=10
  WHEN 3
   player'color(zz):=6
   col:=20
  WHEN 4
   player'color(zz):=4
   col:=32
  ENDCASE
  textcolors(-1,-1,player'color(zz))
ENDPROC format
//
PROC spin'wheel
  pad$:="600!100!300!200!000!"
  wheel'$:="700!200!400!500!200!100"
  wheel'$:+"!400!300!900!000!200!500"
  wheel'$:+"!100!300!200!600"
  wheel'$:=""18""+pad$+wheel'$+"!000!"
  textcolors(-1,-1,13)
  PRINT AT row-4,col: "SPIN"
  PRINT AT row-2,col+2: "
  textcolors(-1,-1,player'color(zz))
  RANDOMIZE
  x:=RND(19,82)
  FOR t:=1 TO x DO
   PRINT AT row-3,col: wheel'$(t:t+4),
   FOR d:=1 TO 75 DO NULL
  ENDFOR t
  PRINT ""146""
  IF wheel'$(x+2)="!" THEN spin'wheel
  get'num
  CASE number OF
  WHEN 0
   total'win(zz):=0; new'win(zz):=0; win(zz):=0
   right:=FALSE
   PRINT AT row+3,col: USING "$####": total'win(zz)
   INPUT AT 20,2,0: "BANKRUPT push [RETURN] twice!": q$
  OTHERWISE
   NULL
  ENDCASE
ENDPROC spin'wheel
//
PROC solve
  textcolors(-1,-1,player'color(zz))
  PRINT AT 20,0: "Enter all spaces & letters!"
  INPUT AT 21,5: "-->": solution$
  IF solution$=remember$ THEN
   PRINT AT 22,3: player$(zz);
   PRINT "solved the puzzle for $",total'win(zz)
   END "Great going"+player$(zz)+"!"9""
```

More ►

```
 ELSE
  PRINT "That is incorrect!!"
  total'win(zz):=0; new'win(zz):=0; win(zz):=0
 ENDIF
ENDPROC solve
//
PROC winnings
 CASE letter$ OF
 WHEN "a","e","i","o","u"
  IF found THEN
   new'win(zz):=found*200
   win(zz):=win(zz)-new'win(zz)
   total'win(zz):+win(zz)
  ELSE
   new'win(zz):=200
   win(zz):=win(zz)-new'win(zz)
   total'win(zz):+win(zz)
  ENDIF
 OTHERWISE
  new'win(zz):=found*number
  win(zz):=win(zz)+new'win(zz)
  total'win(zz):+win(zz)
 ENDCASE
 PRINT AT row+3,col: USING "$####": total'win(zz)
ENDPROC winnings
//
PROC make'choice
 textcolors(-1,-1,player'color(zz))
 PRINT AT 19,1: player$(zz);
 textcolors(-1,-1,15)
 PRINT "Spin [1] or Solve [2]"
 REPEAT
  a$:=inkey$
 UNTIL a$ IN "1,2"
 CASE a$ OF
 WHEN "1"
  spin'wheel
 WHEN "2"
  solve
 ENDCASE
ENDPROC make'choice
textcolors(-1,-1,11)
//
PROC zip'text(row,text$) CLOSED
 tempt$:=text$
 WHILE (LEN(tempt$)<=39) DO
  PRINT AT row,1: tempt$
  tempt$:="<"+tempt$+">"
  FOR d:=1 TO 35 DO NULL
 ENDWHILE
ENDPROC zip'text
//
PROC chk'for'vowel
 IF letter$ IN "aeiou" THEN
  PRINT AT 3,8: "Vowels cost $200 each."
  FOR p:=1 TO LEN(name$) DO
   IF letter$ IN name$ THEN
    total'win(zz):-200
   ENDIF
  ENDFOR p
 ENDIF
ENDPROC chk'for'vowel
//
```

```
PROC print'guess
 IF NOT found THEN
  PRINT AT 4,30: "LETTERS"
  textcolors(-1,-1,10)
  PRINT AT 4,30: "LETTERS"
  PRINT AT 5,30: "GUESSED"
  IF c>39 THEN r:+1; c:=30
  PRINT AT r,c: letter$
  c:+2
 ENDIF
 textcolors(-1,-1,15)
ENDPROC print'guess
//
PROC get'num
 CASE x OF
 WHEN 56,57,58
  number:=0
 WHEN 20,21,22
  number:=700
 WHEN 24,25,26,36,37,38,60,61,62,76,77,78
  number:=200
 WHEN 28,29,30,44,45,46
  number:=400
 WHEN 32,33,34,64,65,66
  number:=500
 WHEN 40,41,42,68,69,70
  number:=100
 WHEN 48,49,50,72,73,74
  number:=300
 WHEN 52,53,54
  number:=900
 WHEN 80,81,82
  number:=600
 OTHERWISE
  number:=x
 ENDCASE
ENDPROC get'num
//
people:
DATA "fireman","school teacher","grocery clerk"
DATA "second baseman","preacher","rock star"
DATA "school bus driver","high school student"
DATA "parking lot attendant","soccer player"
DATA "favorite girlfriend","professional golfer"
DATA "little bo peep","congressman","zoo keeper"
DATA "policeman","taxi driver","frosty the snowman"
DATA "the man in the moon","jack in the box"
thing:
DATA "baseball glove","pair of shoes"
DATA "golf clubs","cat litter","dishwasher"
DATA "thunder storm","computer","boxcar"
DATA "sidewalk","ice cream cone","used car"
DATA "record player","video cassette recorder"
DATA "school books","diamond ring","music stand"
DATA "bath towels","blackboard eraser"
place:
DATA "shopping center","hometown","ski resort"
DATA "golf course","chicago loop","downtown","bed of roses"
DATA "crystal lake","amusement park","junior high school"
DATA "down by the sea shore","office building"
DATA "my sister's room","camp grounds","mountain"
DATA "expressway","family room","by the dock of the bay"
DATA "middle of the road","april in paris"
```

Sets in COMAL 2.0

by Joe Visser and Dick Klingens

In mathematics, sets are used to solve certain discrete problems. In some programming languages (Pascal, ADA) sets are implemented to deal with such problems. It is possible to create sets in COMAL in an easy way.

If an element is a member of a set (is in a set), we can record that with TRUE (one bit); if not with FALSE. For 30 elements we need a sequence of 30 bits, one for each element.

Example:

```
00000000111111...
12345678012345... <-elements
---------------------
00100000011100... <- bit representation of set
```

We have a set with elements:

3, 11, 12, 13, ...

Such a bit sequence can be represented by a real number: a set is a real number. In the following we shall create sets with a maximum of 30 elements.

We need two functions; the first to transfer a set into a sequence of bits (one's and zero's in a string) and the second to transfer a bit sequence into a set.

```
FUNC bstr$(number) CLOSED //set to bits
  DIM binar$ OF 30
  binar$:=bin$(number) // conversion
  WHILE LEN(binar$)<30 DO
    binar$:="0"+binar$ // add leading
  ENDWHILE //                zero's
  RETURN binar$
  //
  FUNC bin$(number) CLOSED
    IF number=0 THEN
      RETURN ""
    ELSE
      RETURN bin$(number DIV 2)+STR$(nu
      mber MOD 2) //wrap line
    ENDIF
  ENDFUNC bin$
ENDFUNC bstr$
//
FUNC bval(binar$) CLOSED
  IF binar$="" THEN
    RETURN 0
  ELSE
    ln:=LEN(binar$)
    RETURN bval(binar$(1:ln-1))*2+VAL(bi
    nar$(ln:ln)) // wrap line
  ENDIF
ENDFUNC bval
```

To create a set of one element we have

```
FUNC setof(element) CLOSED
  IMPORT bstr$,bval
  DIM binar$ OF 30
  binar$:=bstr$(0) // only zero's
  binar$(element):="1"
  RETURN bval(binar$)
ENDFUNC setof
```

Using this function:

```
set1:=setof(2)
set2:=setof(5)
```

we have the sets

(2) and (5)

We can include an element in a set by

```
FUNC include(set,element) CLOSED
  IMPORT bstr$,bval
  DIM binar$ OF 30
  binar$:=bstr$(set)
  binar$(element):="1"
```

More ►

```
  RETURN bval(binar$)
ENDFUNC include
```

And now:

```
  set1:=include(set1,5)
  set2:=include(include(set2,3),6)
```

Then the sets are:

(2,5) and (3,5,6)

With two other functions we can use the set operations union and section.

```
FUNC union(set1,set2) CLOSED
  IMPORT bstr$,bval
  DIM binar1$ OF 30, binar2$ OF 30
  binar1$:=bstr$(set1)
  binar2$:=bstr$(set2)
  FOR t:=1 TO 30 DO
    IF binar2$(t)="1" THEN binar1$(t):="1"
  ENDFOR t
  RETURN bval(binar1$)
ENDFUNC union

FUNC section(set1,set2) CLOSED
  IMPORT bstr$,bval
  DIM binar1$ OF 30, binar2$ OF 30
  binar1$:=bstr$(set1)
  binar2$:=bstr$(set2)
  FOR t:=1 TO 30 DO
    IF NOT (binar1$(t)="1" AND binar2$(
    t)="1" THEN // wrap line
      binar1$:="0"
    ENDIF
  ENDFOR t
  RETURN bval(binar1$)
ENDFUNC section
```

Now:

```
  uni:=union(set1,set2)
  sec:=section(set1,set2)
```

We can show the elements of a set with

```
FUNC elements(set) CLOSED
  IMPORT bstr$
  DIM binar$ OF 30
  binar$:=bstr$(set)
  num:=0 // number of elements
  FOR t:=1 TO 30 DO
   IF binar$(t:t)="1" THEN
     num:+1
     PRINT t; // element
   ENDIF
  ENDFOR t
  PRINT "#",
 RETURN num
ENDFUNC elements
```

Example:

```
  PRINT elements(uni)
  PRINT elements(sec)
```

with output:

```
   2 3 5 6 #4
   5 #1
```

where the number of elements of the set is printed after #.

Creation of the empty set (no elements) is possible with

```
FUNC empty CLOSED
  IMPORT bstr$, bval
  RETURN bval(bstr$(0))
  // or simply:   RETURN 0
ENDFUNC empty
```

A nice example, showing reversed polish notation:

```
set1:=empty
set2:=empty
set1:=include(include(include(set1,3),4),5)
set2:=include(include(include(set2,4),5),6)
PRINT elements(union(section(set1,set2)
,include(set1,7)) // wrap line
```

More ►

Because a real number is represented by 5 bytes, the maximum number of elements of a set is 40 (5 times 8 bits).

If sets with more than 40 elements are wanted, one can use array's. Procedures instead of functions must then be used.

It is also possible to use strings, leaving out the conversion into real numbers.

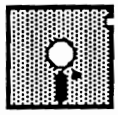
I give some examples in the follwing program:

```
DIM a$ OF 80, b$ OF 80
DIM c$ OF 80, d$ OF 80
a$:=empty$; b$:=empty$
c$:=empty$; d$:=empty$
// 4 sets are now created; each
// with room for 80 elements
//
FUNC empty$ CLOSED
  DIM r$ OF 80
  FOR t:=1 TO 80 DO r$:+"0"
  RETURN r$
ENDFUNC empty$
//
FUNC include$(set$,element) CLOSED
  set$(element:element):="1"
  RETURN set$
ENDFUNC include$
// using this function:
//
a$:=include$(include$(a$,3),4)
b$:=include$(include$(b$,4),5)
b$:=include$(b$,6)
//
FUNC elements(set$) CLOSED
  num:=0
  FOR t:=1 TO 80 DO
    IF set$(t:t)="1" THEN
      PRINT t;
      num:+1
    ENDIF
  ENDFOR t
  PRINT "#",
  RETURN num
ENDFUNC elements
// and it is possible to print the
// elements:
//
PRINT elements(a$)
PRINT elements(b$)
//
FUNC union$(set1$,set2$) CLOSED
  DIM r$ OF 80
  FOR t:=1 TO 80 DO
    IF set2$(t:t)="1" THEN
      set1$(t:t):="1"
    ENDIF
  ENDFOR t
  RETURN set1$
ENDFUNC union$
//
c$:=union$(a$,b$)
PRINT elements(c$)
//
FUNC section$(set1$,set2$) CLOSED
  FOR t:=1 TO 80 DO
    IF NOT (set1$(t:t)="1" AND set2$(t:
    t)="1") THEN // wrap line
      set1$(t:t):="0"
    ENDIF
  ENDFOR t
  RETURN set1$
ENDFUNC section$
//
PRINT elements(section$(a$,c$))
PRINT elements(section$(b$,c$))
```

Now the execution of this program is faster than for a program which includes conversion into real numbers. A second advantage is that the program is more readable. However, there is more memory occupied by strings than by real numbers.

Interested readers may convert real functions such as minus into string functions for set operations. ■

Getting Started



by Colin Thompson

COMAL 0.14 is a disk loaded system. Each recent *Today Disk* includes the complete COMAL 0.14 system, as well as the programs from *COMAL Today*. Other good disks to start with are in the *Paradise Pak* or *COMAL Starter Kit*. You must load the system before you attempt to **RUN** any COMAL programs. LOAD COMAL like this:

1. From BASIC, insert the disk.
2. Type: **load "boot*",8** *<return>*
3. Type: **run** *<return>*
4. While the language loads, read the screen display.
5. Answer "**y**" to the error message prompt.
6. You only have to LOAD the language once. If you are in a "menu" choose the Quit option to drop back into standard COMAL.

GETTING STARTED

COMAL is like BASIC in some ways. It uses the disk drive, screen and printer in much the same way as BASIC. You may LOAD and SAVE programs, look at the disk's directory and LIST programs to the printer. The big difference between the two languages is that COMAL is much smarter than BASIC. The first thing you should do is LOAD a program and RUN it. To find out what programs are on the disk, look at the disk's directory like this:

Press the *<STOP>* key
Type: **cat** *<return>*

This is COMAL's way of showing the directory (catalog). The directory will appear on your screen, but won't erase the program in memory! (Dual drive users can use **cat 0** or **cat 1**). Select a program to LOAD and then type:

load "program name" *<return>*

You don't need to add the usual **",8"** that BASIC requires. COMAL knows about the disk drive. When the red light goes out, type:

run *<return>*

The program you selected will now appear on your screen.

If you want to take advantage of COMAL, you can LOAD and RUN programs automatically like this:

chain "program name" *<return>*

Suffixes (and prefixes) are used in COMAL to identify the type of program or file. Here is a list of commonly used suffixes:

.lst	A LISTed program. Don't LOAD it. Instead type: **new** *<return>* **enter "filename"** *<return>* **run** *<return>*
.txt	A sequential text file
.proc	A procedure listed to disk
.func	A function listed to disk. Procs and funcs are not complete programs. Programmers use them.
.hrg	A hi-res graphic picture
.obj	Pure Machine Language
.dat	A data file

There are many other suffixes, but the general rule is don't LOAD programs that have a suffix. These files are usually recorded last on the disk and are identified as ">--data file--<" (don't load). **Don't LOAD a SEQuential file.**

SOME TIPS

List programs to the printer like this:

select "lp:" *<return>*
list *<return>*

More ►

Note: you cannot list the disk directory to the printer like this. Instead, use the program dir'printer3 on the *Best Of COMAL* disk.

Use the **PASS** keyword to send commands to the drive:

```
pass "n0:new'name,id"
pass "i0"
pass "v0"
pass "d1=0"
```

PASS is the same as (in BASIC):

```
open1,8,15
print#1"<command>"
close1
```

Delete a line number (or range) like this:

```
del 100
del 10-40
del-600
```

Renumber the program with:

```
renum
```

Your journey through COMAL should be enjoyable and exciting. Disk loaded COMAL 0.14 was intended to be a simple learning language. It has proven to be much more, but still performs it's prime purpose better than any other language. It should be your first look at structured programming. ENJOY!

Further References:

Real Help for Real Beginners, COMAL Today #10, page 40
Beginners, COMAL Today #9, page 75
A Simple Exercise for Non Gurus, COMAL Today #7, page 22 ■

Batchfile Cleanup

by Jack Baldrige

Once I learned what batch files were, I realized that they were a beautiful way to exercise the imagination. If someone set up a contest for the best batch files in different categories, there should be some great entries.

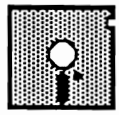
I don't claim that I'd be one of the winners, but I'm sure that I'd be a contestant. I've spent several happy hours dreaming up batch files. This is my favorite:

```
page//
use system
defkey(4,"      "13"") // 7 spaces
renum
delete"0:tem"
list"0:tem"
size
enter"0:tem"
size
delete"0:tem"
//
// Relink Package if Necessary
//
list -20
```

This is a batch file for cleaning up a program name table. I start by deleting my old temporary file before I list the program to disk. This protects me in case the file was still there, and I couldn't save over it. I once lost the latest version of a program when I tried to list it under a name that was already on the disk.

While I'm not about to organize a contest, I feel sure that some of you have favorite batch files too. If so, I'd really like to see them. How about it? ■

Disk Editor



by Phyrne Bacon

Disk'editor can be used with a 1541 or MSD disk drive to edit any sector of any track on a formatted disk.

Disk'editor can be used to unscratch files, to recover from an accidental short new (**pass"n0:name"** without id), or to do any other disk editing function.

GETTING STARTED

Before using *disk'editor*, make a backup copy of the disk. If anything on track 18 is changed, validate the disk afterwards:

Type: **pass"v0"**

After COMAL has been loaded, type new and then type size. If it says, *11838 bytes free*, you have the memory expanded and may load *disk'editor*. If not, you must expand memory first. See the COMALites Unite article in the front of this issue of COMAL Today. All recent versions of the "HI" program expand memory.

LOADING THE FIRST SECTOR

You will be asked to enter the decimal track and sector numbers. That block will be loaded from the disk.

After a sector has been loaded from the disk, the decimal track and sector numbers and *first half* appear near the top of the screen. The 128 bytes of the half-block are divided into 8 columns and 16 rows. The data for the first half-sector (half-block) appears below in two forms. Each byte is given in hex on the left and a character on the right.

The byte 65 (in decimal) would appear as 41 (in hex) on the left and **a** on the right.

Many bytes, 0 for example, are not used to represent letters or numbers, and are represented on the right by . (period). The byte 160 (shift-space) is used to fill out all disk directory filenames to sixteen bytes. It is represented on the left by **a0** and on the right by **+**.

I. EDITING

INFORMATION (i)

Whenever a half-block is displayed on the data screen, you can see the info (help) screen by pressing **i**. The info screen lists all the *disk'editor* commands. Press any key to return to the data screen.

THE CURSORS

There are two cursors which move together. The cursor on the left is on the hex byte (the corresponding decimal appears above). The cursor on the right is on the corresponding character. The position of the byte the cursor is on also appears above. The position number ranges from 0 to 255.

Use the cursor keys to move the cursors. They wrap: when they go over the top edge they appear on the bottom row; when they go over the right edge, they appear in the left column; etc. Press the *<home>* key to return the cursors to the upper left-hand corner.

SWITCHING HALF BLOCKS (h)

To change from the first half-block to the second half, press **h**. To switch back, press **h** again.

ENTERING HEX

To change a byte by entering a hex number, move the cursor to the byte and type the hex number. If you type the first digit

More ►

incorrectly, press an illegal character such as **x** to keep the byte unchanged. To enter a hex number you must type both digits. Example: type **0d** for a carriage return.

ENTERING DECIMAL (n)

To enter a decimal number, press **n**, enter the number, and press *<return>*.

TYPING MODE (t)

To enter typing mode, press **t**. Once you are in typing mode, each key typed will be entered in the byte under the cursor, and the cursor will move to the next byte. The cursor keys cannot be used in the typing mode. To exit typing mode, press *<return>*.

ZEROING A BLOCK (z)

To set a block to zero, press **z**, and answer **y** to the question. This will zero all bytes except for the first two.

II. LOADING AND SAVING

LOADING A BLOCK (k or l)

To load a block using decimal track and sector numbers, press **l** (lower-case L). To load using hex, press **k**. Press *<return>* after each number. If you pressed **l** or **k** by mistake, give a false track or sector number such as **99**.

RELOADING A BLOCK (r)

To reload a block, press **r**. The block with the track/sector numbers on the screen will be reloaded.

SAVING A REVISED BLOCK (s)

To save a revised block to the same track/sector from which it was loaded, press **s**. You will be asked if you are sure. Press **y** to save the block.

WRITING A NEW BLOCK (v or w)

To save a block to a different track/sector, press **w** and enter decimal numbers, or press **v** and enter hex numbers. Press *<return>* after each number. **S**, **w**, or **v** can be used to move a block from one disk to another. If you pressed **w** or **v** by mistake, give a false track or sector number such as **99**.

JUMP TO NEXT BLOCK IN FILE (j)

To move to the next block in the file, press **j**. The first two bytes of a block give the track/sector of the next block in the file.

NEXT BLOCK ON TRACK (^)

To move to the next block on the track, press the *<up-arrow>* key.

JUMP TO CURSOR BLOCK (u)

If a track/sector appears as a pair of hex numbers on the screen, move the cursor to the track (the one on the left) and press **u**. The block will be loaded and the first half will appear on the screen.

PREVIOUS BLOCK (p)

To move to the block loaded previous to the one on the screen, press **p**.

III. AFTERWARDS

EXIT (x)

To exit, press **x**.

VALIDATING THE DISK

Any time the disk directory is edited,

More ►

update the BAM (Block Availability Map).

Type: **pass "v0"**

Then, to check the disk directory type:

cat

This program is on *Today Disk #13.*

Further Reference:

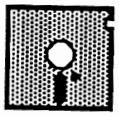
Inside Commodore Dos by R. Immers and G. Neufeld
Fix Disk Errors, COMAL Today #11, page 16
Directory Editor, COMAL Today #8, page 55
Disk Editor, COMAL Today #7, page 56
Disk Editor, COMAL Today #5, page 45
Disk Directory Manipulator, COMAL Today #3, page 6

BLOCKS FREE

Jack Baldridge sent us this COMAL 2.0 function to determine how many blocks are available on a disk.

```
FUNC blocks'free CLOSED
  // by Jack Baldridge
  DIM dummy$ OF 1
  unit':=VAL(UNIT$(:LEN(UNIT$)-1))
  drive:=unit' MOD 2
  unit':=(unit' DIV 2)+8
  TRAP
    MOUNT
    OPEN FILE 79,"u"+STR$(unit')+":$"+
    STR$(drive)+"/s1/t+/d-" // wrap line
    dummy$:=GET$(79,2); blocks#:=0
    FOR x#:=1 TO 35 DO
      sectors:=ORD(GET$(79,4))
      IF x#<>18 THEN blocks#:+sectors
    ENDFOR x#
    CLOSE FILE 79
    RETURN blocks#
  HANDLER
    CLOSE FILE 79
    RETURN -1
  ENDTRAP
ENDFUNC blocks'free
```

Directory Notes



by Phyrne Bacon

Print'directory prints the disk directory normally or with the scratched file entries and the beginning track and sector position of each file included. The blocks free are read from the BAM; so that the blocks free number matches that given by **cat.**

If the sum of the file lengths and the blocks free is not 664, the message *pass"v" may free xx blocks* is printed on the screen. If a file type number is non-standard, it is printed in parentheses in the PRG column.

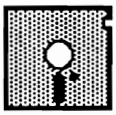
Directory'probe gives more complete directory information than *print'directory.* It can be used with *disk'editor* to help recover disks that were damaged by an unwanted scratch or new command. All these programs are on *Today Disk #13.*

Directory'probe prints the track/sector of each directory block followed by its eight file entries with the beginning track/ sector for each entry. Optionally, the complete disk block chain for each entry and all of the disk block chain fragments may also be printed. A disk block chain is a linked list of track/sector (block) numbers for each block that is used by a file. Fragments are disk block chains that are no longer connected to the directory.

If the disk block chain of a directory entry does not have the length indicated in the directory, the message *wrong length* is printed, which indicates that the file has been overwritten. *This message is also printed for random* (rel) *files, but does not indicate a problem with the disk.*

Scratched files are indicated by --- or 0 in the PRG column. Unused entries are indicated as starting at track 0, sector 0.

Unscratching Files



by Phyrne Bacon

Disk'editor, on *Today Disk #13*, may be used to recover a file which has been accidentally scratched (deleted). **Don't save anything on the disk until the error has been corrected.** First make a backup disk, using a whole disk copier such as four minute backup. Put the original disk in a safe place, and work on the backup.

I. THE DISK DIRECTORY FILE

There are 256 bytes in any disk block. For each block of a file except the last block, the first two bytes are used for the link address of the next block in the file. The first byte is the track and the second byte is the sector of the next block in the file. In the last block of a file, the first two bytes are **00 nn**, where **nn** is the position of the last byte in the file.

The directory file begins at track 18 sector 0. Track 18 sector 0 has the disk name and the BAM (block available memory). The first eight directory entries are in track 18 sector 1. The remaining entries are also in track 18.

If you load 18-0 (track 18 sector 0) into *disk'editor*, you can jump to 18-1 by pressing **j**. Pressing **j** repeatedly will jump you through the whole directory file. To go to the second half of a block, press **h**.

II. DIRECTORY ENTRIES

Each directory block is divided into eight equal entries: four entries in each half block. There are 32 bytes in each entry. In *disk'editor*, each entry is exactly eight columns wide, and four rows long. Each entry is used to store the information about one disk file.

In *disk'editor*, we number the eight columns from one to eight, and the sixteen rows in each half-block from one to sixteen.

The first two bytes in the directory are not used (except in the first entry which uses them to point to the next track and sector of the directory file). The byte in the third column of the first row of any directory entry indicates the file type of the entry:

0	deleted	($00)
128	deleted	($80)
129	SEQuential	($81)
130	PRoGram	($82)
131	USeR	($83)
132	RELative	($84)

Adding 64 ($40) to the file type number gives a **protected file** number. A protected file cannot be scratched. Observe that $82+$40=$c2 in hex; hence $c2 is the type number for a protected program file.

The bytes in columns four and five of the first row of an entry give the track/sector of the first block of the file. If they are **11 00** in hex, (11 hex is 17 decimal) the first block of the file is in 17-0. If you move the *disk'editor* cursor over the **11**, you will see *dec=17* near the top of the screen. If you press **u**, the editor will jump to 17-0 (the track/sector under the cursor).

The sixteen bytes beginning in column six of the first row of the entry and ending in column five of the third row of the entry give the filename padded on the right with shifted-spaces ($a0). The DOS looks for the first shifted space and puts the final quote there. The rest of the name is also printed when the directory is read. It is possible to have a name say: "name",8: or "name"sys 49152. In *disk'editor*, shifted

More ►

spaces are indicated by **a0** on the left and **+** on the right. To type in a filename, move the cursor to column six of the first row of the entry, press **t**, type the name, pad it with shifted-spaces, unlock the shift-lock key if necessary, and press *<return>*.

The next three bytes (starting at row 3 column 6 of each entry) are only used for relative files. Then there are four bytes which are not used and two bytes which are used by the infamous save and replace.

The last two bytes in the entry, in columns seven and eight of the fourth row, are the number of blocks which the file uses. The number is given in lo/hi format (lo+hi*256). Since 12 in hex is 1*16+2=18, 18 blocks would appear in *disk'editor* as **12 00**, with the 12 in column seven and the 00 in column eight. Any file length less than 256 blocks will have 00 in the eighth column. When the cursor is on the 12, you can read the decimal file length as *dec=18* near the top of the *disk'editor* display.

III. PROGRAM BLOCK CHAINS

Say your program is two blocks long. The track/sector of the first block of the program is listed in the disk directory. The track/sector address of the second block (the link address) is given in the first two bytes of the first block. The first two bytes of the second block (the last block of the program) are used to indicate how much of that block is used for storing the program.

IV. UNSCRATCH FILES

In *disk'editor*, a scratched file will have 128 or 0 ($80 or $00) in the third column of the first row of its entry, and in a track/sector *print'directory* printout, or *directory'probe* printout, a scratched file will have --- or (0) in the PRG column. *Directory'probe* will also indicate if the file is the *wrong length*; that is, if the actual chain length does not match the length on the directory because the file has been overwritten.

To unscratch a file, first locate the disk directory block which has the file entry (this is clearly labeled in the printout from *directory'probe* or can be found by reading directory names from *disk'editor*). Load the block into *disk'editor*, find the entry within the block (press **h** to see the second half-block if necessary), and then change the byte in the third column of the first row of the entry to 129 ($81) for a seq (sequential) file, or 130 ($82) for a prg (program) file. To enter hex 81, press **8** and then **1**. To enter decimal 129, press **n**, enter 129, and then press *<return>*. For example, to unscratch the second entry in the directory, move your cursor down to the fifth row in the first half-block of 18-1, and then over to the third column. Change the byte to 81 or 82 (in hex). Then save the block back to the disk by pressing **s**.

V. CORRECT THE BAM

After making the changes, look at the directory listing by typing **cat**. The entries have been restored, but they are not safe yet! The BAM shows the corresponding blocks as not having been used. To correct the BAM, and make the file(s) safe, validate your disk by typing: **pass"v0"** ■

FAST DIR REVISITED FIX

Gerald Hobart notes that the *read'dir* procedure on page 50 of *COMAL Today #12* has two lines reversed in the CASE statement. The lines with rel and usr should be exchanged. ■

Integrated Software 

Several COMAL 2.0 Cartridge users have discovered that the cartridge includes a small integrated system of built-in programs, including Modem, Word Processor, Data Base and Picture Printer.

MODEM TERMINAL

Issue the command:

select output "sp:"

Now all your normal output is directed through a normal Commodore modem. You can even have automatic true ASCII conversion by adding the attribute "/a+":

select output "sp:/a+"

You also can switch from the keyboard as the input location, to the modem using this command:

select input "sp:"

WORD PROCESSOR

We now regularly get letters using the cartridge's built in word processor. This is how it works:

Type anything you want on the screen. Use the *STOP* key to get to the next line. Other commands include:

CONTROL K - Erase to end of line
CONTROL L - Go to the end of the line
CONTROL B - Move back one word
CONTROL F - Move forward one word

Plus, the cursor keys work, as do the *INSERT* and *DELETE* keys.

When the screen looks right, just press *CONTROL P* and the textscreen is printed on your printer.

DATA BASE

James White sent us the following note (using the built in Word Processor mentioned above):

COMAL is great. It even has a built in Data Base. The way it works is with the command **FIND**. If you set up a program file for your database and enter the data as a comment line (//), then **FIND** will seek the string you search for. You can use **DEFKEY** to set the function keys to expedite the search. If you set up portions of the data as a **PROC** you can limit the search to only that portion. Here is an example of how you might set it up. The great thing is that the program doesn't take up much memory, leaving the rest for data.

```
0010 USE system
0020 defkey(1,"find"+CHR$(34))
0030 defkey(3,"find books "+CHR$(34))
0040 //James White 505-982-0567
0050 //Captain COMAL 608-222-4432
0060 PROC books
0070   //Moby Dick  Herman Melville
0080   //Hamlet     William Shakespeare
0090 ENDPROC books
```

PICTURE PRINTER

If you have a Commodore 1525, MPS801, MPS803, or compatible, all you need to do is press *CONTROL D* and the full graphic screen (multi-color or hi-res) is printed on your printer.

Further Reference:

Modem Fun With COMAL 2.0, COMAL Today #9, page 10
Amazing Delete Key, COMAL Today #7, page 19
COMAL 2.0 Auto ASCII Conversion, COMAL Today #6, page 40 ■

Xact Copy

by Patrick L. Roye

The **HARDCOPY** command of COMAL 2.0 is quick and easy to use but only prints out the CHR$ value of the characters on the screen. A **B** on the screen and a **B** on the printer have the same CHR$ and look similar, but the dot pattern of each is different. If you use a custom defined font (such as one from the *Font Disk*), the standard hardcopy will not be able to output the character exactly - only its CHR$, which has not changed.

This is where my new hardcopy comes in. It first checks which font is in effect, locates the text screen, and then it gets a character from the screen, reads its 8x8 bit raster from the proper font, flips it upside down (necessary for my printer) and then finally prints it out, using the hi-res printing mode of the printer. With this new enhanced hardcopy, you can print out an exact copy of the text screen no matter what the characters look like. I hope this will be a help to fellow COMALites.

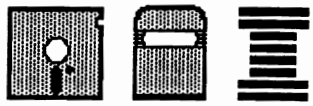
[Editor's note: David Stidolph converted this program to copy the text screen characters to the graphics screen. That way the screen can be printed using the method you usually use to dump a graphics screen - or save the screen for printing later. Both procedures are listed here, and can be merged with other programs as needed. The first, xactcopy, copies the text screen to the printer, while the second, copyscreen, copies the text screen to the graphics screen. Unfortunately, Mr. Roye did not tell us what type of printer he used this procedure with. Can someone let us know what printers his procedure works with?]

Text to Printer

```
PROC xactcopy CLOSED
  USE system
  USE font
  DIM raster$ OF 8
  DIM xraster$ OF 8
  set:=PEEK($d018)
  IF set=23 THEN charset:=3
  IF set=21 THEN charset:=2
  IF set=191 THEN charset:=1
  IF set=189 THEN charset:=0
  raster$(1:8):=""
  xraster$(1:8):=""
  OPEN FILE 78,"u4:/t+/s0",WRITE
  PRINT FILE 78: CHR$(27),CHR$(65),CHR$(8)
  screenloc:=PEEK($0288)*256
  FOR col#:=0 TO 39 DO
    PRINT FILE 78: CHR$(27),CHR$(75),CHR$(200),CHR$(0),
    FOR line#:=24 TO 0 STEP -1 DO
      char:=PEEK(screenloc+line#*40+col#)
      getcharacter(charset,char,raster$)
      FOR flip:=1 TO 8 DO
        xraster$(9-flip):=raster$(flip)
      ENDFOR flip
      PRINT FILE 78: xraster$,
    ENDFOR line#
    PRINT FILE 78: CHR$(10)
  ENDFOR col#
  PRINT FILE 78: CHR$(27),CHR$(64)
  CLOSE FILE 78
ENDPROC xactcopy
```

Text to Graphics Screen

```
PROC copyscreen CLOSED
  USE graphics
  USE font
  DIM raster$ OF 8
  DIM xraster$ OF 8
  set:=PEEK($d018)
  IF set=23 THEN charset:=3
  IF set=21 THEN charset:=2
  IF set=191 THEN charset:=1
  IF set=189 THEN charset:=0
  graphicscreen(0)
  background(0)
  pencolor(1)
  clearscreen
  screen:=PEEK(648)*256
  FOR row:=0 TO 24 DO
    FOR col:=0 TO 39 DO
      char:=PEEK(screen+row*40+col)
      getcharacter(charset,char,raster$)
      FOR lines:=1 TO 8 DO
        location:=$dfff+lines+row*320+col*8
        POKE location,ORD(raster$(lines))
      ENDFOR lines
    ENDFOR col
  ENDFOR row
ENDPROC copyscreen
```

CALC Benchmark 

One of the standard methods of comparing programming languages is the Byte magazine *CALC* Benchmark. Here are the results we came up with:

Time	Error	Language
38.2	**0.00000000**	**COMAL 2.0 fast mode**
53.4	**-0.00000342**	**COMAL 0.14 fast mode**
64.2	0.00000000	C128 BASIC fast mode
72.9	**0.00000000**	**COMAL 2.0**
106.6	**-0.00000342**	**COMAL 0.14**
107.4	0.00000000	C64 BASIC
135.7	0.00000000	C128 BASIC

Since COMAL 0.14 shares many of the calculation routines with C64 BASIC, you will notice that their times are very close. It also is evident that COMAL 2.0 includes some improved routines, putting it on the top of the list. You will also note that FAST mode on the C128 helps a lot with calculations.

This is the COMAL program used:

```
// delete "0:byte5/85calc.lst"
//   list "0:byte5/85calc.lst"
print "byte 5/85 calc benchmark"
poke 162,0 // set jiffy clock
poke 161,0 // to zero for times
poke 53296,3 // c128 fast mode
number#:=5000 // times in loop
a:=2.71828
b:=3.14159266
c:=1
for trial#:=1 to number# do
   c:=c*a
   c:=c*b
   c:=c/a
   c:=c/b
endfor trial#
ending:=peek(162)+peek(161)*256
poke 53296,0 // c128 normal mode
print "done"
print "error =";c-1
print using "###.# sec": ending/60
```

Tom Kuiper was kind enough to run this benchmark test for some languages available for the IBM PC with these results:

Time	Error	Language
7.8	**0.00000000**	**IBM PC COMAL (8087)**
19.7	-4.58 e-13	True BASIC compiled
42.3	**0.00000000**	**IBM PC COMAL**
69.2	0.00000018	IBM PC BASIC
82.6	-0.00000001	Turbo Pascal
91.3	0.00000000	Better BASIC

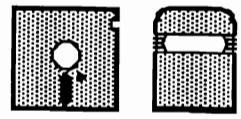
I found it interesting that COMAL does so well compared to the popular IBM PC languages. Even C64 COMAL 2.0 did better than some IBM PC languages!

The nice thing about IBM PC COMAL is that it can make use of the 8087 math chip if it is present. Many of the other languages don't even recognize the chip. Notice how much faster COMAL was when the chip was in the computer!

Here is the BASIC version:

```
10 rem : save "@0:calc-byt5/85-bas",8
20 rem
30 rem fast : rem fast mode c128
40 s=time
50 n=5000 : rem times in loop
60 a=2.71828
70 b=3.14159266
80 c=1
90 for t =1 to n
100 c=c*a
110 c=c*b
120 c=c/a
130 c=c/b
140 next
150 e=time
160 print "done
170 print "error = ";c-1
180 print "time = ";(e-s)/60
190 rem slow : rem normal mode c128 ■
```

1000 Primes Revisited



One standard way of comparing programming languages is by using the prime number sieve program, popularized by *BYTE* magazine. Our version finds the first 1000 prime numbers with these results:

Time	Language - Not printing
19.4	**COMAL 2.0 fast mode**
41.2	**COMAL 2.0**
50.8	**COMAL 0.14 fast mode**
94.5	C128 BASIC fast mode
101.1	**COMAL 0.14**
137.4	C64 BASIC
199.8	C128 BASIC

Time	Language - Print each prime
29.2	**COMAL 2.0 fast mode**
54.5	**COMAL 2.0**
103.1	C128 BASIC fast mode
115.1	**COMAL 0.14**
151.9	C64 BASIC
215.1	C128 BASIC

Here is the COMAL program:

```
// use c128
// select output "u7:" // 80 column screen
// poke 53296,3 // c128 fast mode
poke 162,0 // jiffy clock set
poke 161,0 // to zero for times
si#:=3962; count#:=0
dim flags#(0:si#)
for i#:=0 to si# do
  if not flags#(i#) then
    prime#:=i#+i#+3
    k#:=i#+prime#
    count#:+1
    // print prime#; // <<<<optional
    while k#<=si# do
      flags#(k#):=true
      k#:+prime#
    endwhile
  endif
endfor i#
ending:=peek(162)+peek(161)*256
//poke 53296,0 // c128 normal mode
print
print "count=";count#
print using "###.# sec": ending/60
select output "ds:"
```

Since both BASIC and COMAL automatically fill a dimensioned array with 0 - the initialize array section in *BYTE*'s sieve is skipped. This sieve test is for the first 1000 **prime** numbers. *BYTE*'s test checks the first 14,003 numbers. This test program is also different than the one used by *The Midnite Gazette* (see Kevin Quiggle's benchmark test page in *COMAL Today #12*, page 37).

The test was run twice for each language, once printing the prime numbers as they were found. Then again without the printing. The difference in the two times is the time it takes for the langauge to print results on the screen.

We printed the results of the *BYTE* sieve test program in *COMAL Today #5* on page 1. (Yes, COMAL did very well!)

Here is the BASIC program:

```
10 s=time
20 sz%=3962 : c%=0
30 dim f1%(sz%)
40 for i=0 to sz%
50 if f1%(i)<>0  then 210
60 pr%=i +i +3
65 print pr%;
70 k%=i +pr%
80 if k%>sz% then 120
90 f1%(k%)=1
100 k%=k%+pr%
110 goto 80
120 c%=c%+1
210 next i
300 e=time
310 print "count=";c%
320 print "time=";(e-s)/60 ■
```

Super Chip Primes

by Captain COMAL

One of the 100 new commands in Super Chip is **PRIME**. It can tell you if any number between 0 and 65535 is a prime number in much less than 1 second. It's only natural to be curious; I wanted to see how long it would take for 1000 primes. Of course, it is at a disadvantage here since for each number tested it starts from scratch (it doesn't save any of its previous work). Even so, it did quite well. We used this program:

```
use system2
use math
if host$="c128" then
  use c128
  turbo(true); keypad(false)
  select output "u7:" // 80 column screen
endif
start:=time; count#:=0; num#:=1
while count#<1000 DO
  num#:+2 // skip even numbers
  if prime(num#) then
    count#:+1
    //print num#;
  endif
endwhile
ending:=time-start
if host$="c128" then turbo(false); keypad
(true) //wrap line
print
print "Count=";count#;"Last prime=";num#
print using "###.# seconds": ending/60
select output "ds:"
```

These are the results:

Time	Version
23.1	c128 not printing
32.6	c128 printing
47.9	c64 not printing
61.2	c64 printing ■

Best Selling Books

Introduction to Computer Programming is the best selling book for June, the month that we released it! It is designed to be the first book in a three book series. If anyone is interested in writing either the Intermediate or Advanced books, please let us know. We also are pleased that *COMAL Workbook* is in the charts. Gordon Shigley, its author, lives just a couple miles from our office!

April 1986

#1 - COMAL From A To Z
by Borge Christensen
#2 - COMAL Handbook
by Len Lindsay
#3 - COMAL Workbook
by Gordon Shigley
#4 - Foundations With COMAL
by John Kelly
#5 - Cartridge Graphics & Sound
by Captain COMAL's Friends

May 1986

#1 - COMAL From A To Z
by Borge Christensen
#2 - COMAL 2.0 Packages
by Jesse Knight
#3 - COMAL Handbook
by Len Lindsay
#4 - COMAL Workbook
by Gordon Shigley
#5 - COMAL Yesterday
First Four COMAL Today issues

June 1986 (thru June 27)

#1 - Introduction to Computer Programming
by J William Leary
#2 - Introduction's Answer Book
by J William Leary
#3 - Cartridge Tutorial Binder
by Frank Bason & Leo Hojsholt
#4 - COMAL Handbook
by Len Lindsay
#5 - COMAL Workbook
by Gordon Shigley ■

Behind the Scenes

Introduction To Computer Programming Book Set

by J William Leary

I was introduced to COMAL two summers ago at Earlham College. It made *"sense!"* I taught COMAL to a second-year computer class as an intro to Pascal. The more I worked with it, the more I realized BASIC had to go; it has. So this year all six of my classes have COMAL. Next year, when Pascal is reintroduced, COMAL will be a prerequisite.

The frustration I had last year was with teaching materials. The books that came closest to my needs were Roy Atherton's *"Structured Programming With Comal"* and John Kelly's *"Foundations in Computer Studies with COMAL."* Atherton's material was too advanced and, in addition, had (along with Kelly's) two major *"short comings":*

1. They were written for pupils who have better academic backgrounds than their American counter parts.

2. They lacked a list of *"objectives"* for each chapter.

Neither of these is a criticism--(in a strict sense it might be more of an indictment of our school program here). Therefore, I prepared my own material. The book seemed a natural extension of it all. A few book-specific observations:

1. Flow charts or structure diagrams (as Atherton calls them) are not included.

Having taught for 35 years I feel I have a "*sense*" of the American student. Because of the mix, in terms of backgrounds, I have found by actual in-class experience that these become an end in themselves; instead of adding clarity, they cause confusion to all but the best of students. Therefore, I chose not to include them in a beginning text.

2. The answers to the problems found at the end of each chapter are included as a continuous printout. I really don't know exactly how to handle them in terms of page numbering. *[They are a separate Answer Book.]*

Some further thinking:

I see a three-book series for COMAL. The first is a beginning text; the second an intermediate text; and, the third, an advanced one.

Pascal doesn't have this. Oh, there are texts that would fit the above three categories but there is no one series to provide intelligent articulation.

With the spectacular growth (quite understandable!) of COMAL, I feel this would be an excellent project. Teachers need this kind of a series. So do those who are not in school, but are now getting into programming.

A series would eliminate needless redundancy and would provide the user with a high degree of program sophistication.

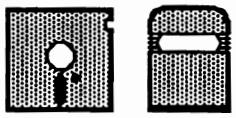
[The book has just been published and is the number one best seller it's first month out! Congratulations!] ■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

COMAL TODAY - THE INDEX

It's almost ready now. A full index to everything in *COMAL Today*, issues 1 through 12. Kevin Quiggle is compiling it. We hope to have all entries on disk as well as in a book of about 64 pages! ■

COMAL Comments



by Sol Katz

One of the really nice things about COMAL (among the many) are the built-in disk operating commands. To start, you don't need to tack on ",8" after each disk command, nor is it necessary to open a channel to the drive before sending commands. Saving a file is done by:

save"0:filename"

This is similar to BASIC. Scratching a file is much easier, though. Just use:

delete "0:filename"

If you want to send the output to a printer, type:

select output "lp:"

The "lp:" means line printer. When you want the output back on the screen, type:

select output "ds:"

The "ds:" means data screen.

The other disk commands are not as convenient in COMAL 0.14 as in COMAL 2.0, but are at least as easy as in BASIC. The PASS command "passes" instructions to the disk drive. Some of the more common disk instructions are implemented as follows:

disk command	COMAL 0.14 statement
New	pass "n0:diskname,id"
Copy	pass "c0:newfile=0:oldfile"
Rename	pass "r0:newname=0:oldname"
Initialize	pass "i0"
Validate	pass "v0"

And now for some discussion about LIST and EDIT when changing a file. LIST will show the automatic indenting that makes this such a great programming language. However, there is a minor problem when you want to change a line that includes a PRINT statement longer than 40 characters! Use EDIT in this case. Otherwise, the automatic indenting puts the indented spaces permanently into the line!

To have the program wait while a message is displayed, try:

```
print "hit any key to continue."
while key$=chr$(0) do null
```

Finally, a hint for those just starting with COMAL. To help differentiate between COMAL keywords and user defined procedures, type:

setexec+

This will print the word EXEC in front of each procedure call. ■

Expert Systems

by Michael Erskine

Man: *Computer.*
Machine: Ready.
Man: *Translate, English to Korean,"The quick red fox jumped over the lazy brown dog's back."*
Machine: Word "the": there are 24 main uses, 7,800 characters of data.
Machine: Word "quick": there are 11 main uses, 6 adjectives and 5 nouns, 1340 characters of data.
Machine: Word "red": there are 10 main uses; 1 noun, 1 adjective, 2 intransitive verbs, and 2 transitive verbs, 600 characters of data.
Man: *Stop.*
Machine: Ready.
Man: *Get to the point.*
Machine: There is a 1 in 6.3 times 10 to the eleventh power chance of producing a correct Korean translation of the English sentence "The quick red fox jumped over the lazy brown dog's back," on the first attempt. Even after the first attempt, there is no way to know which choices made were correct. The first attempt will require reading 26,360 characters of data.
Man: *What is necessary?*
Machine: I need to know the grammatical structure of any language I shall be required to translate.
Man: *That's a pretty tall order. You must know there are substantial differences between even closely related languages like English and Spanish.*
Machine: The data is required to help establish the functions of the words to be translated.
Man: *Explain.*
Machine: If I don't know that a verb in Korean is always predicated, I will translate verbs as adverbs and adverbs as verbs or gerunds.
Man: *OK, we'll give you the grammar.*
Machine: That is a fine start. I'll also need constant updating of my store of definitions (including slang, surnames, and any untranslatable concepts), a method of correcting the grammar and spelling errors of the input data, and a way to identify and ignore or translate those nonsensical noises you humans make (such as zzz for snoring or @*#%& for profanity).
Man: *Why do you want constant updating? A dictionary is a dictionary!*
Machine: I need updating for the same reason as your dictionary needs updating, languages evolve with the society in which they are used. A dictionary for American English frequently shows different spelling and usage from a dictionary for British English.
Man: *OK, We'll get data entry right on it.*
Machine: Wait, there's more. I'll be needing the ability to compare the word I'm currently translating with the surrounding context and to further compare the sentence under processing with the document context to ensure consistency of the logical flow of meaning in the whole translation....
Man: *Stop.*
Machine: Ready.
Man: *Won't this take a lot of memory?*
Machine: Certainly, but you get what you pay for.

More ►

Man: *Well, I suppose if humans can make these translations, a super machine like a Cray can also. How much storage does a man have?*

Machine: Various estimates from medical researchers suggest about 1.2 billion locations of direct access RAM. Of course, there is also the off line storage of libraries, data bases, and other people.

Man: *Define locations?*

Machine: That cannot be defined. There are those who suggest one location can hold all the information contained in a painting for example. It is really an unmeasured quantity, but certainly much more than 8 bits.

Man: *Humans have unmeasurable storage.*

Machine: What is more, it takes years to teach a human to be a competent translator, even if he is immersed in the learning of the language. Which, by the way, means learning the customs and ideology of the people who use the target language.

Man: *OK. Purchase more memory, 30 billion bytes would be sufficient. Build the data base. Write rules of word usage. Write rules of grammar. Make the decisions necessary to translate. Compare and retranslate as required. Finally produce the final document...*

Machine: Stop.

Man: *Ready.*

Machine: There's just one more thing, and it's most important. When a translator is working, it always is necessary to arrange the output in a manner which is esthetically pleasing, humanly logical, and syntactically satisfying. He must carry over from one language to another nuances of respect (both implied and expressed) and other emotional implications contained in the original language. A translator must be creative.

Man: *Word "creative": there is 1 main use, adjective, adverb or noun, 120 characters of data. Creative equals originative.*

Man: *Word "originate": there are 2 main uses, 2 verbs, 160 characters of data...*

Machine: Stop.

Man: *Ready.*

Machine: List "originate."

Man: *Input Korean or English?.*

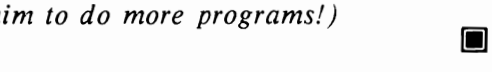
Machine: Korean.

Man: *Originate:to begin, to arise from.*

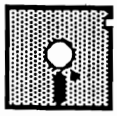
Machine: You need an update. Add to the existing files for originate, originative, create, and creative, a new entry in each file.

Man: *Ready.*

A year ago Michael told us that COMAL needed a powerful program so good that people would get COMAL just so they could run it. So he wrote an expert system, **PROTO-D.** *Systems like this usually sell for hundreds of dollars. All Michael wants is a donation from those who use it - a start of COMAL ShareWare. The set of programs (Editor, Reader, and Writer) will be on ShareWare Disk #1 (due July 1986). If you appreciate his efforts, please send him a contribution (currently he's delivering pizza and mowing lawns - convince him to do more programs!)*



BASIC to COMAL



by Sol Katz

Breathes there a hacker
with soul so dead,
who in his own heart hath not said,
"There has got to be some easy way to convert all those BASIC programs to COMAL."

(apologies to poetry lovers everywhere.)

Two COMAL 0.14 programs comprise the *basic2comal* system. This two part system is designed to help programmers convert C64 BASIC programs to COMAL. Before you get too excited, it only does about 80% of the text conversion and none of the conversion necessary to get rid of the tortured logic inherent in BASIC "spaghetti" code.

To run the programs, the following condition must be met:

Both COMAL programs and the BASIC program to be converted must be on the same disk with enough free disk space to write several files.

The *basic2comal* system does the following:

* converts BASIC tokens to keywords
* puts spaces around keywords
* converts GOTO # to GOTO GT# (a label) and then puts GT#: in the right place;
* converts GOSUB # to EX# (a proc name) and puts the proc name in the right place
* strips BASIC line numbers and puts in sequential COMAL line numbers
* converts ON..GOTO and ON..GOSUB to CASE structure
* inserts a GOTO in IF expression THEN #
* converts user defined BASIC functions (DEF FN) to COMAL format
* partially converts LEFT$, MID$ and RIGHT$ to COMAL string format, using the LEN function and a dummy string variable, XX$
* converts RETURN to ENDPROC
* converts all IF..THEN statements to IF..THEN..ENDIF block format
* inserts some commas in some PRINT statements (BASIC isn't as strict as COMAL when separating variables from functions like TAB and SPC)
* converts REM to //
* converts NEXT to ENDFOR
* hides BASIC keywords that have no COMAL equivalents
* appends the string "OF 10" to all DIM variable$(number) statements
* splits all multiple statements on one line to separate lines
* several other minor conversions

The program leaves the following for the user to do:

* DIM string variables that are not BASIC arrays
* convert OPEN statements to COMAL format
* initialize numeric variables before they are used in logic statements
* insert occasional commas in print statements
* substitute the correct string variable in the converted LEFT$, MID$ and RIGHT$ functions
* figure out how to get the SPC function (converted to SPCS) to work
* figure out how to get SYS #,# to work in COMAL (an undocumented BASIC feature that showed up in some public domain programs from Commodore)
* convert TI, TI$, VAL, GET and other functions to COMAL functions (this will be much easier in COMAL 2.0 then in COMAL 0.14)
* change PEEK and POKE to work with COMAL addresses

More ►

* put quotes around strings in some DATA statements
* and last but most important, untangle the logic that uses subroutine line numbers as targets for GOTO statements

Basic2comal asks for the name of the BASIC program file to be translated. It writes output to two files after deleting older versions. The first is called *structured'basic*. It contains an ASCII file of the converted BASIC program (including the original BASIC line numbers as comments). The second file (called *gotos'and'procs*) contains the line numbers that were part of GOTO and GOSUB statements. The GOSUB line numbers were made negative to overcome the C64 limitation of allowing a maximum of 3 files open at once (the BASIC file, *structured'basic*, and *gotos'and'procs*). I had to store the line numbers in a file and split the program into 2 parts to overcome the 12K available memory limit in COMAL 0.14.

The first program then chains the second. It reads the line numbers into 2 arrays, the positive numbers into the **goto array** and the negative numbers, after convertion to positive, into the **proc array**. The arrays are then sorted. It then opens *structured'basic*, compares the sorted GOTO #s and GOSUB #s with the BASIC line numbers and inserts them as LABELs or PROC names in the correct order. When LABELs and PROC names occur at the same BASIC line number, the PROC name is entered first. The BASIC line numbers are striped off and new line numbers added. Then each line is written to a file called *almost'comal*.

The last step in the automated conversion is to find (and hopefully correct) the remaining syntax errors. I used the dynamic keyboard technique to put the commands:

new
enter "almost'comal"

on the screen, and POKEd returns (CHR$(13)) into the keyboard buffer. This automatically clears memory and feeds each line of ASCII code to COMAL's syntax checking routines. It is as if you entered the line invisibly at the keyboard. If the line is correct it goes on to the next line. If the line has an error, the line and standard COMAL error message will be displayed. You **must** deal with the syntax error. If you don't know what's wrong, make the whole line a comment by putting a ! right after the line number (a ! is coverted to // by COMAL for you). If you use the *up* or *down* cursor keys, the program will stop being **entered** at the previous line. If you accidentally stop the automatic **entering**, you can start over by typing:

enter "almost'comal"

After the last line has been automatically **entered** from the file, you have a COMAL program that is ready for debugging. I suggest you save it under a new name since the file *almost'comal* is deleted when *basic2comal* is run again.

The rest is up to you. Good luck and don't forget to send copies of your converted programs to COMAL Users Group, U.S.A., Limited so that we can all share them.

[This system provides an excellent starting point for anyone wishing to improve on it. Please let us know if you do!]

More ►

```
// delete "0:basic2comal-p1"
//  by sol katz
// save  "0:basic2comal-p1"
//
// convert basic to comal - part 1
//  by sol katz, december 1985
// colorado comodore computer club
//
new'line'number:=10
gt'mat:=0
ex'mat:=0
disk'get'init
//
dim case'token$ of 15
dim name$ of 20
dim in'line(255)
dim out$ of 80, number$ of 8
//
print chr$(147)
print "BASIC to COMAL - Part 1"
print
print "Please enter name of BASIC program"
print "to translate to COMAL."
print
input "basic file to translate : ": name$
infile:=2
//
open file infile,name$,read
print name$," status ",status$
delete "0:goto'and'proc"
open file 5,"goto'and'proc",write
load'tokens
//
delete "0:structured'basic"
open file 4,"structured'basic",write
file'end:=false
disk'get'skip(2,infile,file'end)
//
get'line
while not file'end do
 clear'flags
 decode
 print'line
 get'line
endwhile
clear'flags
close
print "Pass 1 complete - calling"
print """basic2comal-2"" to finish translation"
chain "basic2comal-p2"
end
//
proc clear'flags
 quote:=0; lef:=0; rit:=0; mid:=0
 case':=0; colon:=0; dim':=0
 comma':=0; paren:=0
 rem':=0; com:=0; if':=0; dollar:=0
 func':=0; input':=0; goto':=0; then':=0
 if gt'mat or ex'mat then write'goto'or'proc
endproc clear'flags
//
proc get'line
 get'line'number
 start:=1
 count:=0; rtn:=0
 repeat
  count:+1
  ascii:=disk'get(infile,file'end)
  in'line(count):=ascii
 until ascii=0 or file'end
 line'end:=count
 quote:=0
endproc get'line
//
proc get'line'number
 disk'get'skip(2,infile,file'end)
 ascii:=disk'get(infile,file'end)
 lo'byte:=ascii
 ascii:=disk'get(infile,file'end)
 hi'byte:=ascii
 line'number:=hi'byte*256+lo'byte
endproc get'line'number
//
proc decode
 posit:=1
 repeat
  ascii:=in'line(posit)
  if rem' or quote then
   move'character
   if ascii=34 then quote:=0
  elif ascii>=128 and ascii<=218 then
   if ascii=137 and if' then print'line
   if then' then
    print'line
    then':=0
   endif
   if ascii=178 and func' then ascii:=165
   decode'token
  else
   case ascii of
   when 34 // quote
    quote:=1
   when 37
    ascii:=35 //int
   when 58 //:
    ascii:=32
    print'line
    colon':=1
    if case' then
     out$:=out$+"otherwise"
     print'line
    endif
    if func' then
     out$:=out$+"endfunc"
     print'line
    endif
    clear'flags
   when 36 //$
    if dim' then dollar:=1
   when 41 //)
    if rit then
     out$:=out$+":len(xx$)"
     rit:=0
    endif
    move'character
    ascii:=32
    if func' then
     print'line
     out$:=out$+" return "
    else
     if dim' and dollar then out$:=out$+" of 10"
     dollar:=0
    endif
   when 59 //;
    if input' then ascii:=58
   when 40
    if lef or rit or mid then ascii:=32
   when 44 //,
    if gt'mat or ex'mat then write'goto'or'proc
    if case' then
     print'line
     case'num:+1
     out$:="when "+chr$(case'num+48)
     print'line
     out$:=case'token$
     ascii:=32
    elif com=1 then
     ascii:=58
     com:=0
    elif com=2 then
     com:=0
    elif mid then
     ascii:=40
     mid:=0
     com:=1
    elif lef then
     ascii:=32
     out$:=out$+"(1:"
     lef:=0
    elif rit then
     out$:=out$+"(len(xx$)+1-"
     ascii:=32
     com:=2
    endif
   otherwise
    if then' then
     print'line
     then':=0
     case ascii of
     when 48,49,50,51,52,53,54,55,56,57 // number
      out$:=out$+" goto gt"
      gt'mat:=1
     otherwise
     endcase
    endif
   endcase
   if paren and ascii<>44 then
    out$:=out$+","
    paren:=0
   endif
   if ascii=34 then
    if out$(len(out$))=")" then out$:=out$+","
   endif
   if ascii<>32 then
    move'character
    if comma' and ascii=41 then paren:=1
    if gt'mat then number$:=number$+chr$(ascii)
    if ex'mat then number$:=number$+chr$(ascii)
   endif
```

More ►

```
 endif
 posit:+1
until posit=line'end or file'end
endproc decode
//
proc move'character
 out$:=out$+chr$(ascii)
 if comma' and ascii=41 then paren:=1
endproc move'character
//
proc print'line
 new'line'number:+5
 print new'line'number;out$;
 //print file 4: new'line'number,out$;
 write file 4: line'number,out$
 if quote then
  print chr$(34);
 endif
 // if start then
 //  print file 4: "//",line'number
 print "//",line'number
 out$:=""; quote:=0
endproc print'line
//
proc set'flags
 case ascii of
 when 139
  if':=1
  rtn:=1
 when 167
  then':=1
 when 137
  goto':=1
  gt'mat:=1
 when 141
  ex'mat:=1
 when 145
  case':=1
 when 133
  input':=1
 when 150
  func':=1
 when 134
  dim':=1
 when 143
  rem':=1
 when 200
  lef:=1
 when 201
  rit:=1
 when 202
  mid:=1
 when 166,163 //spc and tab
  comma':=1
  if not out$(len(out$)) in " ," then out$
  :=out$+","  // wrap line
 otherwise
 endcase
endproc set'flags
//
func disk'get(file'num,ref file'end) closed
 poke 2026,file'num
 sys 2025
 file'end:=peek(144)
 return peek(2024) //value of character
endfunc disk'get
//
proc disk'get'init closed
 for loc#:=2024 to 2039 do
  read v
  poke loc#,v
 endfor loc#
 data 0,162,0,32,198,255,32,207
 data 255,141,232,7,32,204,255,96
endproc disk'get'init
//
proc disk'get'skip(count,file'num,ref file'end) closed
 for x#:=1 to count do y:=disk'get(file'num,file'end)
endproc disk'get'skip
//
proc load'tokens
 dim token$(128:218) of 15
 for i:=128 to 218 do
  read token$(i)
 endfor i
 data " end "," for "," endfor "
 data "data "," input file "," input "," dim "
 data " read "," ","  goto gt"," run "," if "
 data " restore "," ex"," endproc"," // "," stop "
 data "case "," wait "," load "," save "," verify "
 data " func "," poke "," print file "," print "," cont "
 data " list ","//clr ","//cmd "," sys "," open file "
 data " close ","//get "," new "," tab("," to "
 data " "," spcs("," then "," not "," step "
 data "+","-","*","/","
 data " and "," or ",">","=","<"
 data "sgn","int","abs","usr","size "
 data "pos","sqr","rnd","log","exp"
 data "cos","sin","tan","atn","peek "
 data "len","str$","val","asc","chr$"
 data "","","","" endif","endcase "
 data "otherwise"," null","207 ","208 ","209 ","210 "
 data "211  ","212  ","213 "," 214  ","215  "
 data "216  ","217  ","218 "
endproc load'tokens
//
proc decode'token
 set'flags
 if case' then
  case'num:=1
  case'token$:=token$(ascii)
  if goto' or ascii=141 then
   print'line
   out$:=out$+"when 1"
   print'line
  endif
 endif
 if ascii=142 and rtn then
  out$:=out$+" return"
 else
  out$:=out$+token$(ascii)
 endif
 case ascii of
 when 139 // if
  line'end:+2
  in'line(line'end-2):=58 //:
  in'line(line'end-1):=203 //endif
 when 145 // on = case
  line'end:+2
  in'line(line'end-2):=58 //:
  in'line(line'end-1):=204 //endcase
 otherwise
 endcase
endproc decode'token
//
func val(s$) closed
 leng:=len(s$)
 ones:=ord(s$(leng))-48
 if leng=1 then
  return ones
 else
  return ones+val(s$(1:leng-1))*10
 endif
endfunc val
//
proc write'goto'or'proc
 if gt'mat then
  write file 5: val(number$)
  gt'mat:=0
 elif ex'mat then
  write file 5: val(number$)*(-1)
  ex'mat:=0
 endif
 number$:=""
endproc write'goto'or'proc
```

End of first program.
Second program is on
the following page.

More ►

```
// delete "0:basic2comal-p2"
//  by sol katz
// save  "0:basic2comal-p2"
//basic to comal  part 2
// to be chained from part 1
// by sol katz, colorado commodore
//     computer club, dec 1985
last:=200; first:=1
dim goto'(last), file'name$ of 15
dim out$ of 80, proc'(last)
dim ans$ of 1
file'name$:="goto'and'proc"
//
get'array
if g'inc>1 then
 quick'sort(1,g'inc,1,g'inc,goto')
 print
 print "sorted goto labels"
 print
 printit(g'inc,goto')
endif
print
if p'inc>1 then
 quick'sort(1,p'inc,1,p'inc,proc')
 print "sorted proc names "
 print
 printit(p'inc,proc')
endif
get'structured'basic
//
print chr$(147)
print "The file ALMOST'COMAL will now"
print "be entered. If there are any syntax"
print "errors you must correct them as if"
print "you had just typed them in and then"
print "hit the return key. If you can't"
print "fix the problem, make the line a"
print "comment by putting a // after the"
print "line number. Then fix it later."
print "Using the UP or DOWN cursor keys will"
print "stop the program!!!"
print
print "Hit any key to start ENTERing program."
while key$<>chr$(0) do null
while key$=chr$(0) do null
print chr$(147),chr$(17),chr$(17),
print chr$(17),chr$(17),"new"
print chr$(17),chr$(17),"enter";chr$(34),
print "almost comal",chr$(34),chr$(19),
poke 631,13
poke 632,13
poke 633,13
poke 634,13
poke 635,13
poke 198,5
end
//
proc swap(i,j,ref nums())
 temp:=nums(i)
 nums(i):=nums(j)
 nums(j):=temp
endproc swap
//
proc quick'sort(first,last,i,j,ref nums())
 dividing'line:=nums((first+last) div 2)
 repeat
  while nums(i)<dividing'line do i:+1
  while nums(j)>dividing'line do j:-1
  if i<=j then
   swap(i,j,nums)
   i:+1
   j:-1
  endif
 until i>j
 if first<j then quick'sort(first,j,first,j,nums)
 if i<last then quick'sort(i,last,i,last,nums)
endproc quick'sort
//
proc printit(number,ref nums())
 for i:=1 to number do
  print nums(i)
 endfor i
endproc printit
//
proc get'array
 open file 5,"goto'and'proc",read
 print status$
 g'inc:=0; p'inc:=0
 while not eof(5) do
  read file 5: basic'line
  if basic'line>0 then
   g'inc:+1
   goto'(g'inc):=basic'line
   print g'inc;" ",basic'line
  else
   p'inc:+1
   proc'(p'inc):=basic'line*(-1)
   print "        ";p'inc;" ",basic'line
  endif
 endwhile
 close file (5)
endproc get'array
//
proc get'structured'basic
 open file 4,"structured'basic",read
 delete "0:almost'comal"
 open file 5,"almost'comal",write
 line:=10; gt'inc:=1; pr'inc:=1
 while not eof(4) do
  read file 4: num,out$
  print num;"  ";out$
  if goto'(gt'inc)=num then
   print file 5: line;"gt",goto'(gt'inc),":"
   line:+10
   while goto'(gt'inc)=num do gt'inc:+1
  endif
  if proc'(pr'inc)=num then
   print file 5: line;"proc ex",proc'(pr'inc)
   while proc'(pr'inc)=num do pr'inc:+1
   line:+10
  endif
  print file 5: line;out$
  line:+10
 endwhile
 close
endproc get'structured'basic
```

Example BASIC to COMAL

Original BASIC program:

```
10 a=1:rem assign 1 to a
20 fora=1to20:printa:next
30 ifa>0thengosub70:goto50
40 print"Shouldn't be printed"
50 print"Past line 40 to 50!"
60 end
70 if a>0 then print"a>0"
80 return
```

After conversion, almost'comal contains:

```
10 a=1
20  //  assign 1 to a
30  for a=1 to 20
40  print a
50  endfor
60  if a>0 then
70  ex70
80  goto gt50
90  endif
100  print "Shouldn't be printed"
110 gt50:
120  print "Past line 40 to 50!"
130  end
140 proc ex70
150  if a>0 then
160  print "a>0"
170  endif
180  endproc
```

Listed in COMAL it becomes:

```
0010 a:=1
0020 //  assign 1 to a
0030 for a:=1 to 20 do
0040  print a
0050 endfor
0060 if a>0 then
0070  ex70
0080  goto gt50
0090 endif
0100 print "Shouldn't be printed"
0110 gt50:
0120 print "Past line 40 to 50!"
0130 end
0140 proc ex70
0150  if a>0 then
0160   print "a>0"
0170  endif
0180 endproc
```

Protection

by Harald Nendza

I wish to be counted as vigorously opposing the practice of putting protected programs on *Today disks*. To suggest that making minor modifications could seriously affect the dependability of a program is really silly. I guess by the statement at the end of the article *Make Your Program A Package, COMAL Today #9*, that someone thinks there is a large group of people whose only joy in life is to make minor but improper changes to programs, and then complain that they don't work. Ain't so. Besides, wiping out the program listing won't stop anyone from disassembling the code and making modifications. It only takes a bit longer to do.

I very well understand that you at *COMAL Today* are following the wishes of those same wonderful people who write these programs. However, I think that you should argue that leaving programs open will help more then it will hurt. Besides, I haven't gone after programmers with a rolled up copy of *COMAL Today* and hit them on the nose because their program didn't work after I took out half the lines. You know, most of us out here are fairly intelligent and may even have some common sense.

With COMAL still new to so many people, why keep the *tricks* hidden among the few who have time to study and play with the language?

What brought about this strong response was my experience a couple of months ago, when I was just starting to get our COMAL SIG organized. I thought that getting the name/address lists into a data base would be a good idea. Well, what better language to use than COMAL. The data base program on the *Best of COMAL* disk seemed to be the way to go for a cheap data base. Then I got *Today Disk #8* with a COMAL 2.0 data base and naturally decided to use that.

Without looking at the code itself, I proceeded to run the program. I entered a couple of names and addresses and decided that it wasn't bad, but I wanted more flexibility in printing to my antique micro-line 82 printer. I had a file of 100+ names and addresses on another computer system that I wanted to get into the program. I had previously entered the data twice into other data base programs that didn't work. I was going to be smarter this time and let the program itself do the hard part. (You may like to type the same thing over and over again, but not me). It should only take a couple of minor changes in the input routine to get and modify data from the modem port. Hour or two at the most, right? ... Wrong!

I did a list to see what the program looked like and got the *source protected* message. Don't tell me someone protected this program and then gave it away. The brain kicks in and reminds me that I've been seeing mention of program protection and I should check the article for the program. There it was. The program is protected because it would just overwelm and confuse me. What to do next? Should I type in all my information and accept whatever the program spit out? Could be a real waste of time. I could find out how the data needed to be stored and write a program to store my own data in the same way. Got a better way? Let's just scrap this program and search for another, or put something together the way I want it. Has anyone heard the one about the people reinventing wheels... It's too bad, the program could have been just what I needed, if only I could have made one minor change. Now, both the program and programmer's name are forgotten. ▣

Keyword Printer

by David Zavitz

Have you ever wondered how COMAL 0.14 kept track of all the built in commands and keywords? Or maybe you wished you could get a complete list of them. This very short program will scan through COMAL's name table and print all the commands and keywords as well as the memory location each starts at.

Further Reference:

COMAL 0.14 Memory Map, COMAL Today #6, page 28
Another Look At COMAL 0.14 Tokens, COMAL Today #6, page 76
See Program Name Table, COMAL Today #5, page 46
How COMAL Statements Are Stored, COMAL Today #5, page 12

```
Starting address:
^       Number of bytes
^     ^ Keyword/Command
^     ^ ^
2069  3 abs
2073  3 and
2077  3 atn
2081  4 auto
2086  5 basic
2092  4 case
2097  5 chain
2103  3 chr
2107  5 close
2113  6 closed
2120  3 con
2124  3 cos
2128  4 data
2133  5 debug
2139  3 del
2143  3 dim
2147  3 div
2151  2 do
2154  4 elif
2159  4 else
2164  3 end
2168  7 endcase
2176  5 endif
2182  7 endproc
2190  8 endwhile
2199  5 enter
2205  3 eod
2209  3 eof
2213  3 esc
2217  4 exec
2222  3 exp
2226  5 false
2232  3 for
2236  4 goto
2241  2 if
2244  2 in
2247  5 input
2253  3 int
2257  5 label
2263  3 len
2267  3 let
2271  4 list
2276  4 load
2281  3 log
2285  3 mod
```

```
Starting address:
^       Number of bytes
^     ^ Keyword/Command
^     ^ ^
2289  3 new
2293  6 endfor
2300  3 not
2304  2 of
2307  4 open
2312  2 or
2315  3 ord
2319  9 otherwise
2329  6 output
2336  5 print
2342  4 proc
2347  4 read
2352  3 ref
2356  3 rem
2360  5 renum
2366  6 repeat
2373  7 restore
2381  3 rnd
2385  3 run
2389  4 save
2394  6 select
2401  3 sgn
2405  3 sin
2409  4 size
2414  3 spc
2418  3 sqr
2422  6 status
2429  4 step
2434  4 stop
2439  3 tab
2443  3 tan
2447  4 then
2452  4 time
2457  2 to
2460  4 trap
2465  4 true
2470  5 until
2476  5 using
2482  4 when
2487  5 while
2493  5 write
2499  4 zone
2504  4 edit
2509  3 cat
2513  6 append
```

```
Starting address:
^       Number of bytes
^     ^ Keyword/Command
^     ^ ^
2520  6 random
2527  4 unit
2532  4 file
2537  4 poke
2542  4 peek
2547  1 chr$(0)
2549  1 chr$(0)
2551  1 chr$(0)
2553  1 chr$(0)
2555  1 chr$(0)
2557  1 chr$(0)
2559  2 //
2562  1 !
2564  2 :=
2567  2 :+
2570  2 :-
2573  1 :
2575  1 ;
2577  1 +
2579  1 +
2581  1 -
2583  1 -
2585  1 &
2587  1 .
2589  1 (
2591  1 )
2593  1 ,
2595  1 *
2597  1 /
2599  1 ^
2601  1 <
2603  2 <=
2606  2 <>
2609  1 =
2611  1 >
2613  2 >=
2616  1 #
2618  1 chr$(13)
2620  3 sys
2624  7 setexec
2632  6 setmsg
2639  6 return
2646  4 func
```

```
Starting address:
^       Number of bytes
^     ^ Keyword/Command
^     ^ ^
2651  7 endfunc
2659  4 null
2664  4 pass
2669  6 delete
2676 10 setgraphic
2687  7 settext
2695  5 frame
2701  4 plot
2706  8 pencolor
2715 10 background
2726  8 plottext
2735  5 clear
2741  6 define
2748  8 identify
2757 11 spritecolor
2769  9 spritepos
2779 10 spritesize
2790 15 spritecollision
2806 13 datacollision
2820 10 hidesprite
2831  3 key
2835 10 spriteback
2846  6 moveto
2853  6 drawto
2860  8 getcolor
2869  4 fill
2874  7 forward
2882  5 setxy
2888  4 left
2893 10 setheading
2904  8 priority
2913  5 penup
2919  7 pendown
2927  6 border
2934  5 right
2940  4 back
2945 11 splitscreen
2957 10 showturtle
2968 10 turtlesize
2979  4 next
2984  8 linefeed
2993 10 fullscreen
3004 10 hideturtle
3015  4 home
```

More ►

Version 1 - horizontal columns

```
dim reply$ of 1
print chr$(147),chr$(14),
print "Keywords printer"
print
input "Printer/Screen (p/s): ": reply$
if reply$="p" or reply$="P" then
 open file 255,"",unit 4,7,write
 select output "lp:"
endif
print
print "Comal 0.14 Keywords &";
print "Command Memory Map"
print "======================",
print "=================="
print "starting address:"
print "^    number of bytes"
print "^    ^  keyword/command"
print "^    ^ ^"
print
print'keywords
select output "ds:"
//
proc print'keywords
 for cell:=2068 to 3014 do
  byte:=peek(cell)
  print cell+1;byte;
  for character:=cell+1 to cell+byte do
   if peek(character)>32 then
    print chr$(peek(character)),
   else
    print "chr$(",peek(character),")",
   endif
  endfor character
  cell:+byte
  zone 20
  print ,
  zone 0
 endfor cell
 print
endproc print'keywords
```

Version 2 - vertical columns

```
num'words:=177
dim reply$ of 1
dim word$(num'words) of 15
dim address#(num'words)
dim count#(num'words)
print chr$(147),chr$(14),
print "Keywords printer"
print
input "Printer/Screen (p/s): ": reply$
print
print "Please wait 20 seconds..."
get'keywords
if reply$="p" or reply$="P" then
 print
 print "Turn on printer and advace"
 print "paper to top of page"
 print
 input "Press RETURN when ready: ": reply$
 open file 255,"",unit 4,7,write
 select output "lp:"
 zone 25
 num'col:=3; header'printer
else
 print chr$(147),
 zone 0
 num'col:=1; header'screen
endif
per'col:=num'words/num'col
print'keywords
select output "ds:"
//
proc print'keywords
 for x:=1 to per'col do
  for col:=0 to num'col-1 do
   num:=x+col*per'col
   print address#(num);
   print using "##": count#(num);
   print word$(num);
   if col<num'col-1 then print ,
  endfor col
  print
 endfor x
endproc print'keywords
//
proc get'keywords
 addr:=2068
 for x:=1 to num'words do
  cnt:=peek(addr); address#(x):=addr
  count#(x):=cnt
  for c:=1 to cnt do
   if peek(addr+c)>31 then
    word$(x):=word$(x)+chr$(peek(addr+c))
   else
    word$(x):="chr$("+chr$(peek(addr+c)+ord("0"))+")"
   endif
  endfor c
  addr:+cnt+1
 endfor x
endproc get'keywords
//
proc header'printer
 print "Comal 0.14 Keywords and";
 print "Command Memory Map"
 print
 print "Starting address",
 print "      Number of bytes",
 print "       Keyword/Command"
 print "^","      ^","       ^"
endproc header'printer
//
proc header'screen
 print "Comal 0.14 Keywords &";
 print "Command Memory Map"
 print "Starting address"
 print "^     Number of bytes"
 print "^     ^ Keyword/Command"
 print "^     ^ ^"
endproc header'screen
```

Power Supply Blues

"The C64 power supply is notorious for its unreliability. One PCG member replaced the Commodore power supply 3 times in 6 months." - Pittsburgh Commodore Group Newsletter January 1986.

"While many things can and do go wrong with your C64 (quality control is a dirty phrase at CBM) one of the most problematic components is the C64 power supply." - NYCig News November 1985.

"The power pack is the weakest link in the computer." - The Guide April 1986.

"I had just received my brand new SFD-1001 drive and BusCard II and couldn't wait to use it. After about 15 minutes of copying programs, my C64 crashed... I found out from three independent sources that I was having power supply problems." - The Guide November 1985.

"...to save money the power supply has no tolerance at its upper limits." - NYCig News November 1985.

Welcome to the Power Supply Blues. If you add so much as a joystick to your C128 or C64 you may have problems with your power supply. The problem is so common, that selling replacement power supplies is a thriving business. How do you know if you are a victim of power supply failure? In the NEPACC News, S. Pellerite gives 5 warning signals to be aware of:

1) A scrambled video output after warm-up of the unit (anything from 10 minutes to 5 or 6 hours).
2) The scrambled video coincides with a computer lockup, of course.
3) Smoke coming from a power supply.
4) Smoke coming from the operator's ears.
5) A smashed keyboard (operator's hand still in it).

Unfortunately, none of us at the COMAL Users Group are knowledgeable about power supplies. But the problem is a serious one, especially for those adding a COMAL 2.0 cartridge to their system. On page 76 last issue we reprinted a good article about power supplies. We still are waiting for Commodore to inform us about the power requirements of the COMAL 2.0 cartridge. Meanwhile, one user provides us with this information:

A power supply should be "driven" at no more than 50% of its maximum load. The longer the power supply is **hot**, the more it "wears down". When it reaches its threshold, its output drops drastically. A bare C64 computer requires .792 amps of power. The Commodore power supply is a 1.2 amp unit. That puts it over the limit before anything is added! A power requirement estimate for the COMAL 2.0 cart is .2 amps. While the cartridge is plugged in, it is drawing power, even if the computer is in "BASIC" mode (unless a cartridge expander is used that allows you to switch several cartridges on/off).

Another user (Roger Walen) tells us that the C128 power supplies made in West Germany are rated at 2.5 DC, while those made in Japan have a 4.3 rating. He advises you to look at the bottom of the power supply for its rating. He was able to exchange his C128 computer for one that worked with his COMAL cartridge.

PROTECT YOUR SUPPLY

* Cool off your power supply. Place small fan next to it and blow air over the surface.
* Never put it on a rug or enclosed area.
* Plug the supply into a switched power strip and turn off the strip when not using the computer. If left "plugged

More ►

in", the power supply remains "on" all the time.
* Keep it out of sunlight or any other heat source.
* Protect it from power "surges" or "spikes".

We may have found a good replacement for the C64 power supply at a very reasonable price. It is the **Maxtron PS-01,** distributed by Chelsea Computer Accessories. Charles Santos told us, *"The Maxtron power supply that I sell is $29.95 plus $2 UPS shipping. You can see by its heavy duty rating that it can drive the COMAL 2.0 cartridge very nicely. I've sold 70 of them in the past year and have not had one returned."* If you need a good power supply for your C64, try the Maxtron. Here are its specs:

Input: 117V, 60HZ, 40watt
Output: 5V DC, @ 1.7 amps
9V AC, @ 1.0 amps

The C128 also has a marginal power supply, but its odd square type connector poses a problem. Charles Santos is trying to find a source of this connector, but has not had any luck. He thinks it may be a European part.

C64 POWER SUPPLY SOURCES

* Chelsea Computer Accessories, PO Box 346, Culver City, CA 90232-0346, phone 213-398-0913. Maxtron price $29.95 + $2, 1.7 amps.
* Computer Place, 23914 Crenshaw Blvd, Torrance, CA 90505, phone 213-325-4754. Price $39.95 + $?.
* Replacement Power Supply, 60 E Main St, Alliance OH 44601, phone 800-821-1297. Price $69.95 + $2, servicable, 1.5 amps, 4 extra 110 V outlets, Master On/Off switch, surge/ spike protected.
* Tenex Computer Express, PO Box 6578, South Bend, IN 46660, phone 219-259-7051. Price $49.95 + $5.75, 1.5 amps, 2 extra 110 V outlets, surge protected, On/Off switch. Manufactured by Huff. Also distributed by Allegro Tech and Universal Software.

Further References:

Power Supply Notes, COMAL Today #12, page 76
Diagnosing Your Sick Commodore, The Guide April 1986, page 15
C64 Power Supplies, NEPACC News Vol 7 No 2
Power Supply Blues, NYCig News November 1985, page 23
Power Pack 64, PCG Newsletter January 1986, page 6
Potpourri, The Guide November 1985, page 60
Questions and Answers, COMAL Today #9, page 6
Power Supply Problems, COMAL Today #6, page 35 ■

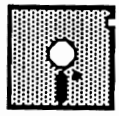
LABELS

Labels have two important functions. One (COMAL 2.0 only) is to RESTORE data statements to any desired line for reading:

```
readme:
DATA "Read me first"
RESTORE readme
READ r$
```

The other use for labels is for comments. A label will never be executed so it won't interfere with a running program. Also, a label comment is listed differently than a standard // comment, which is indented along with the program structure. A label is never indented. It can thus be used to quickly locate an important part of a program. ■

Cute Cubes



by Oren Hasson

[reprinted from Catalina Commodore Computer Club newsletter, April 1986]

COMAL is the best product for your money for the C64 or C128. There is no doubt about it, as you can get it from most local user groups. It was rated "A" for "overall rating", "ease of use", "documentation", "reliability", "error handling" and "value for money" categories by the *Book of Commodore 64 Software, 1985* (published by Arrays, Inc., the book division). It was rated 10 for performance and 9 for ease of use and for reliability by the book *The Best VIC/C64 Software*. No other language got a rating of 10 in any category.

Yes, it is indeed a computer language, and if you find yourself in one of the following categories, it is the language for Commodore computers. If you occasionally write programs for your own use (for business, statistics or fun); if you want to write programs, but are intimidated by the complexity of BASIC (especially on the C64); if you want to write programs that others will be able to read, or want to be able to read and understand programs of others; if you are not satisfied with BASIC, but will never find the time to learn or write programs in ML; if you use computers for teaching, or just want to have fun with easy computing, COMAL is for you. I know, I have been there.

My way to COMAL was easy. Logo and Forth were my first steps beyond BASIC. I found their structure interesting. However, I could not find the time to study Forth thoroughly (the floating point routines were my breaking point - about half way through the book), as it is a time-consuming task. Logo seemed to be unnecessarily complicated (except for its graphics). I also spent a couple of days with Promal (which is powerful, but too complicated, being designed for the real whiz, for whom hexadecimal numbers are the natural way of counting), and tried Pilot. I have also studied FORTRAN, on other computers. Then a friend gave me a disk with COMAL 0.14. After two weeks, I was hooked.

It is not very different from BASIC, so I felt comfortable. The library of the CCCC user group has plenty of programs, so I had something to start with. I found a good starting book, *Starting with COMAL*, by Ingvar Gratte, so I could learn fast.

How come it took me so long to find it? Well, not because I didn't know that it was available. I even saw a demonstration once, but I was tired of medium-quality languages, and did not want to spend more time on another one. I knew nothing about it's structure. This structure is what makes it so easy to write and read. Add to that, powerful commands, a nice environment, easy control of files, and superb graphics, and you know why I was hooked. However, for some reason, no one around me was using it (in contrast, in five countries in Europe, COMAL is the first language taught in school)!

Instead of describing to you the language and its structure, I shall demonstrate just one of its neat features: the procedure. I shall do that by writing and explaining a short program that draws a small cube on the floor of a bigger one.

A procedure, by the way, is a set of commands, like a subroutine, that you call from your main program. *"Big deal"*, you probably say, *"we have subroutines in*

More ►

BASIC". Well, there is a difference. Procedures are called by names that you assign to them, which make the program more readable. But much more importantly, when you call a procedure, you can also send variables for it to use. This will be clarified by the following program.

If you have COMAL 0.14 and you want to enter the program, load it first (there is a fast loader available). Now, quit the "hi" program and type **NEW**, just like in BASIC. Then type **AUTO**. This will supply the line numbers automatically for you, with increments of 10. Press *<return>* at the end of each line, until the whole program is entered, and press *<return>* when you don't want to enter another line. This will stop the **AUTO** command. If you want to list the program, type **LIST**, or **EDIT**. See what the difference is. How do you like that! The space bar freezes the listing, while *<run/stop>* stops it.

Now we are ready for the program itself. Note that two slashes are used for comments (non-executable lines or remarks). **SETGRAPHIC** 0 sets the screen to a high resolution screen (you can change the 0 to 1 for a multi-color screen). The **HIDETURTLE** command hides the turtle (the sprite which is used as a pen). You can put two slashes before this command, if you'd rather see the turtle in action (or just change it to **SHOWTURTLE**).

After preparing the screen, we are now able to draw the cubes. So we write:

cube(100,100,0,80,80,80)

The first three numbers give the cooordinates in 3D of one of the corners of the cube, and the next three send the three dimensions of the cube (change the numbers and see what I mean). Accordingly, the next **cube** command:

cube(130,100,20,30,30,30)

has a starting point of 30 pixels to the right of the last cube, at the same level (bottom of the first), and 20 pixels toward us. Its size, as you can see, is smaller than that of the first cube, and hence it fits nicely in it.

"Hmmm..." may say some of the suspicious among you, *"so COMAL has a feature called cube built-in, which is nice, but not great"*. Well, COMAL doesn't have it, but you can easily construct a cube or a pyramid if you like from the simple commands **MOVETO** and **DRAWTO** that some of the enhanced versions of BASIC have (but which are not as easy to use to construct a program).

You start building a procedure by writing **PROC** and then the name you want to assign to it. In parentheses, you write the type of variables you would like it to receive when you call it. The built-in command, **MOVETO**, accepts two variables, one for the horizontal axis and one for the vertical one, respectively. Now here comes the trick that transforms it to three dimensions. To make the illusion of 3D, you stretch the horizontal axis (x) to the left, and the vertical axis (y) down by the use of:

moveto z*a+x,z*a+y

z is the depth of the third dimension. Very simple and very easy. Now you close the definition of the procedure by writing **ENDPROC**. You don't have to type the name of the procedure at this stage. COMAL is very forgiving. If you list your program, you'll find that COMAL put it there for you (once the program has been run).

More ►

Similarly, when you assign the value -.5 to **a**:

```
a:=-.5
```

you can skip the colon, and write only the "equal" sign:

```
a=-.5
```

Guess what will happen when you list the program? Good, you learn fast (I told you this was easy...).

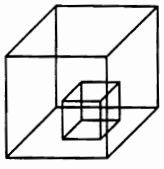
In a very similar manner, you define **drawto3d** by using the built-in **DRAWTO** command. Now, I don't have to explain what **line3d** does. If you understand that, you are ready for COMAL. **Cube** is the final procedure, which uses the other procedures that we just constructed.

You can see now how this modularity of the language makes things easy for you. When you're done, this is the time to explore. You can change cube sizes, separate them from each other, transfer the small cube to the roof to the other, etc. Try the **line3d** procedure. You'll like it. Also, try to write your own procedures. In COMAL, you can execute procedures in the direct mode, within the program, or within other procedures. **Have fun!**

I have used the graphics of COMAL only to introduce you to procedures. Graphics were not the reason I got interested in COMAL, although I like them a lot. I haven't even scratched the surface of COMAL, and I haven't talked about the cartridge. I cannot avoid the temptation of saying a few words about it too. The cartridge is not cheap, but is extremely powerful. (It adds 64k ROM to your computer). Yes, there is also a version for the IBM, completely compatible with the Commodore version, and soon there will be a version for the Apple (again, compatible).

I had some thoughts about buying a Commodore 128. I don't now. Except for the memory, the option of CP/M, and the drive speed, my SX-64 is now more powerful, and at least in some operations, even faster. Some say COMAL 2.0 (the cartridge) is better than Pascal (not only on the Commodore). Undoubtedly, it is easier to use.

```
// save "cubes"
// now comes the main program
//
setgraphic 0
hideturtle
cube(100,100,0,80,80,80) // call for fist cube
cube(130,100,20,30,30,30) // call for second cube
//
// below are the procedures (subroutines of COMAL)
//
proc moveto3d(x,y,z)
 moveto z*a+x,z*a+y
endproc moveto3d
//
proc drawto3d(x,y,z)
 drawto z*a+x,z*a+y
endproc drawto3d
//
proc line3d(x0,y0,z0,x1,y1,z1)
 moveto3d(x0,y0,z0)
 drawto3d(x1,y1,z1)
endproc line3d
//
proc cube(x0,y0,z0,long,high,deep)
 l:=long; h:=high; d:=deep; a:=-.5
 line3d(x0,y0,z0,x0,y0+h,z0)
 drawto3d(x0,y0+h,z0+d)
 drawto3d(x0,y0,z0+d)
 drawto3d(x0,y0,z0)
 drawto3d(x0+l,y0,z0)
 drawto3d(x0+l,y0+h,z0)
 drawto3d(x0+l,y0+h,z0+d)
 drawto3d(x0+l,y0,z0+d)
 drawto3d(x0+l,y0,z0)
 line3d(x0,y0+h,z0,x0+l,y0+h,z0)
 line3d(x0,y0+h,z0+d,x0+l,y0+h,z0+d)
 line3d(x0,y0,z0+d,x0+l,y0,z0+d)
endproc cube
```



Encrypt - Easy Reader

ENCRYPT

by Joel Rea

This program allows you to encrypt and decrypt sequential and program files. You provide the source file and a key. It creates a new file. The key may be any non-zero real number. Each number results in a different encryption. If you encrypt a file and forget the key, **no one** can recover the file.

To decrypt the file, simply encrypt it again with the same key. You can get more security by using multiple passes, each with different keys. You then must decrypt using the same keys **in reverse order**. So, if you encrypt once using *555.1212*, and again using *-9876.54*, you must decrypt using *-9876.54* and then *555.1212*.

More ►

```
// delete "encrypt file"
// save  "encrypt file"
//  by Joel Ellis Rea
DIM infile$ OF 32, outfile$ OF 32
PAGE
PRINT "File Encryptor -- by Joel Ellis Rea"
PRINT "WARNING -- If you forget your key and"
PRINT "lose your unencrypted original, NO ONE"
PRINT "can recover your file!"
PRINT
PRINT "Your key may be any positive, non-zero,"
PRINT "number. Each number produces a different",
PRINT "encryption! Any PRG, SEQ or USR file"
PRINT "may be encrypted. To be safe, this"
PRINT "program will only encrypt to a COPY of"
PRINT "the file! It will NOT encrypt ""in place""!"
PRINT
//
// get encryption code
REPEAT
 INPUT AT 16,0,15: "Key:  ": key'
UNTIL key'>0
//
// get and open INPUT file
REPEAT
 INPUT AT 18,0,40: "Input file:  ": infile$
 TRAP
  OPEN FILE 8,infile$,READ
  ok:=TRUE
 HANDLER
  PRINT ERRTEXT$
  ok:=FALSE
 ENDTRAP
UNTIL ok
//
// get and open OUTPUT file
REPEAT
 INPUT AT 20,0,40: "Output file:  ": outfile$
 TRAP
  OPEN FILE 9,outfile$,WRITE
  ok:=TRUE
 HANDLER
  PRINT ERRTEXT$
  ok:=FALSE
 ENDTRAP
UNTIL ok
//
// now encrypt the file
WHILE NOT EOF(8) DO
 PRINT FILE 9: encrypt$(GET$(8,254),key'),
 key':=0
ENDWHILE
CLOSE
END "Done."
//
FUNC encrypt$(strg$,key') CLOSED
 IF key' THEN RANDOMIZE key'
 FOR i#:=1 TO LEN(strg$) DO
  strg$(i#):=CHR$(ORD(strg$(i#)) BITXOR RND(0,255))
 ENDFOR i#
 RETURN strg$
ENDFUNC encrypt$
```

EASY READER

by Joel Rea

Easyread displays sequential text files to the screen in a way that makes them **very** easy to read. This is done by printing them on the **graphic** screen with two extra rows of pixels between each line of text. This gives 20 text lines per screen. Descenders of one line don't touch the tops of capital letters in the line below.

The program is self-documenting and has a powerful directory feature: when asked for a filename, hit *<return>* for a listing of all sequential files on the current UNIT. If you want a more specific directory listing, just precede a standard filename (using wildcards) with a "$". "$1:t*=s" would give all sequential files beginning with "t" on drive 1 of unit 8. By itself, "$" means **all** files on the current UNIT. This program is on *Today Disk #13.* ■

Code Doctor

by Richard Bain

Coming from a Pascal background, I quickly fell in love with COMAL, but missed a few a few of Pascal's features. One of them was the ability to use procedures and functions as parameters in other procedures and functions. (For the rest of this article, I will use **procedure** to mean procedures and functions). This feature is particularly useful in procedures such as a graph drawing procedure which can take any function as input. In COMAL 2.0 there are now two ways to use procedures as parameters.

The easiest way to do this is to use external procedures. The external definition allows the use of a string variable to name the file containing the procedure to be called. If this variable is a parameter, then the external procedure can be replaced by a new one every time the calling procedure is used.

```
f$:="0:func.name"
graph(f$)
//
proc graph (f$)
  func graph'me(x) external f$
  y:=graph'me(0)
  // put rest of proc here
endproc graph
```

This method has a few drawbacks. Loading a procedure from disk takes time, particularly if it is loaded several times. Also, the external procedure itself must be closed and it can't use **IMPORT**. These restrictions limit the usefulness of external procedures.

The second method is similar to the first in that a string variable parameter is used to pass the name of one procedure to another one. This method uses my package called code'doctor. Code'doctor allows open and closed procedures to be passed. It is fast since no disk access is needed. To use code'doctor successfully, several steps must strictly be followed:

```
0010 call'procs("print'proc1")
0020 a$:="print'proc2"
0030 call'procs(a$)
0040 //
0050 proc print'proc1
0060   print "proc1"
0070 endproc print'proc1
0080 //
0090 proc print'proc2
0100   print "proc2"
0110 endproc print'proc2
0120 //
0130 proc call'procs(passed'proc$) closed
0140   use code'doctor
0150   pass'proc("passed'proc")
0160   exec passed'proc
0170 endproc call'procs
```

Lines 10-30 are used to call the main procedure, call'procs, and to pass the names of other procedures to it. Note that the name of the other procedure may be contained in a string variable or constant. If those procedures have parameter lists, the parentheses would not be used with their names. The $ and # are also not included.

Lines 50-110 contain the code for the procedures being passed to call'procs.

Lines 130-170 contain the code of the main procedure. Line 130 is the header. Passed'proc$ is the string value parameter which receives the name of the procedure being passed to this one. Passed'proc$ can not be a **REF** parameter or an error will result. Lines 140-150 are used to convert the variable, passed'proc$, from a string value parameter to a procedure name. They **must** be the first two lines of the procedure. Note that "passed'proc", **in quotes**, is the name of the string value

More ►

parameter in line 130 as well as the name of the procedure call in line 160. Passed'proc$ no longer exists, but the passed procedure can now be called. If a function were being passed instead of a procedure, pass'proc would still be used in the same way.

The above example is intended for demonstration purposes only. A useful program which uses code'doctor to graph functions is included on *Today Disk #13*. It should be enough (along with the above demo program) to let you use code'doctor as long as you have used procedures before.

Code'doctor is not perfect and has at least one major flaw. It is related to nested procedures with repeated use of the same names. Code'doctor also has an interesting side affect. It may **IMPORT** the passed procedure using its global name. For example, print'proc1 would be **imported** into call'procs in line 10 of the example above. This allows recursion under COMAL's dynamic scope rules.

I learned a lot about COMAL's internal structure while working on code'doctor. This included new insights about the name table, the two stacks, and dynamic scope rules. If I get enough feedback about this package, I may discuss some of this in future articles.

Further References:

How to Use the META Package, COMAL Today #10, page 65
COMAL 2.0 External Procedures, COMAL Today #7, page 27
Pass an Array as a Parameter, COMAL Today #4, page 48
Parameters; Function as Parameter; Local and Global, COMAL Today #2, page 32 ■

Notes

EXTERNAL PROCEDURES

David Martin of Ames, Iowa writes: a valuable feature of many high level languages is the ability to pass a function or procedure name into a subprogram. COMAL provides this quality with the IMPORT statement. However, using EXTERNAL can be more powerful because the name of an EXTERNAL Procedure or Function can be a string variable. This is a very powerful feature of the language, as it can be used to write dynamic or self modifying programs. A short example:

```
INPUT"procedure to execute:":name$
EXEC example
PROC example EXTERNAL name$
```

This program will execute any appropriate procedure which had been saved to the disk. A string variable can be passed into a function or procedure which, in turn, can make an EXTERNAL call corresponding to the string passed in. Unfortunately, EXTERNAL calls do not first check to see if the module requested is already in memory so that repetitive calls are slow due to disk access. I have been unable to figure out how to load from memory, rather than from disk, in order to circumvent this problem and would appreciate hearing from anyone with ideas along this line.

COMAL HELP

Future Technical Institute has been providing education in electronics and computer technology since 1980. They provide applications, sales, support, installation, and hands on workshops. They now are supporting Mytech COMAL. They can be contacted at 31834 Village Center Road, Westlake Village, California 91361, phone: (818) 706-2229. ■

Significant Discussion

by Kevin Quiggle 

Yesterday, I drove my rusty Datsun into the gas station and filled it up with 9.5 gallons of gas. A quick check of the odometer showed that I had traveled 141.9 miles since the last fill-up. Since I don't believe human beings should be forced to go through the mental drudgery of arithmetic, I drove home and calculated my gas mileage on my computer. Quickly dividing 141.9 miles by 9.5 gallons, I got an answer of 14.936842 mpg.

Now, it's entirely possible that my rusty Datsun gets exactly 14.93642 miles per gallon, but the numbers I used to come up with that figure do not give me any right to calculate out to six decimal places. Just how many decimal places do I have the right to claim? If I say my car gets 14.9 mpg, is that any more correct than saying it gets 14.936842 mpg? In fact, 14.9 mpg is not any more correct than 14.936842 mpg, because I have not used the proper number of *"significant"* digits in my answer.

What makes one digit significant and another one not? The technical answer is that a number is made up of significant digits when all of the digits in it but the last one are true, and the last one is in doubt. For example, my odometer measured 141.9 miles. No odometer is perfectly accurate, but most are accurate to at least the nearest mile, so it may be that I actually drove 141.8 miles, or 142.0 miles. The last digit, the one that measures tenths of a mile, is a bit shaky, so my miles number has four significant digits (the last doubtful digit is still *"significant"*).

The rules for determining significant digits are as follows:

1) Trailing zeros after a decimal point are significant.
2) Zeros before a decimal point are only significant if they have non-zero digits in front of them.
3) Zeros after a decimal point which are followed by non-zero numbers are only significant if there are non-zero numbers in front of the decimal point.
4) All non-zero digits are significant.

All of those rules can be confusing, so here are some examples to illustrate them:

Sig Digits	Number
5	123.45
6	123.005
1	0.005
3	12300
5	12300.
6	123.450

Not allowing for the proper number of significant digits in a calculated result is a common failing, especially when people use computers. It's not very easy for computers to handle this problem, because internally all numbers are stored with the same number of significant digits, no matter how many digits were entered originally. Usually, people just round off their numbers to a specified number of decimal places, and let it go at that. The only real way around this problem is to enter the number as a character string, calculate the number of significant digits, and then convert the number string to a number. Since calculating the number of

More ►

significant digits is the hard part, printed below is a COMAL program which shows one way to do it (this program will run with either version of COMAL). The program will ask you for a number, and then print the number of significant digits in the number you entered. This is a very simple example, so it does not check to make sure that what you enter is actually a number.

Once you know how to calculate significant digits, you need to know one more rule in making calculations: When you make a calculation, the result should have the same number of significant digits used to make the calculation. So my actual mileage was 141.9 miles (four significant digits) divided by 9.5 gallons (two significant digits) to give 15 mpg (two significant digits).

```
// delete "0:sigdig"
// save   "0:sigdig"
//  by kevin quiggle
dim num$ of 10
print "significant digits calculator"
repeat
  input "enter number (0 to stop): ": num$
  if num$<>"0" then print sigdig(num$)
until num$="0"
//
func sigdig(n$) closed
  while n$(1)="0" do n$:=n$(2:len(n$))
  dec'pt:="." in n$
  if dec'pt then
    n$:=n$(1:dec'pt-1)+n$(dec'pt+1:len(n$))
    while n$(1)="0" do n$:=n$(2:len(n$))
  else
    while n$(len(n$))="0" do n$=n$(1:len(n$)-1)
  endif
  return len(n$)
endfunc sigdig
```

Directory Fix

Starting with *Today Disk #9* we have been using *David's Directory Designer* to put comments in our disk directories. Recently, we found a bug in that program which affects our disks. The problem is that as new directory entries were added, new directory blocks were being used. However, the Block Availability Map (BAM) was not updated to show this. This problem is on *Today Disks #9, #10, #11*, and all other disks we released with *0 block USR* files. This problem doesn't affect any of the programs running from the original disk, but may show up on a copy of the disk.

If your copy program only copies the disk blocks which have been allocated, then the entire directory won't be on the new disk. You won't be able to find or use all the programs and files.

For those wishing to back up the *Today Disks* using the BASIC backup program included on many of our disks, adding this line will give a correct backup:

```
2625 if t%=18 then for j=1 to 8:bm%(18,s-j)=0:next
```

For those with access to a copy program which copies the entire disk (independent of the BAM), just copy the disk normally.

After using either copy method, issue one more command to your **newly copied** disk:

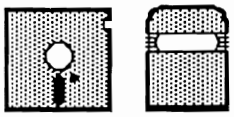
```
pass "v0"
```

You now have an error free disk. We regret any inconvenience this problem may have caused.

Further Reference:

Backup A Disk, COMAL Today #10, page 60

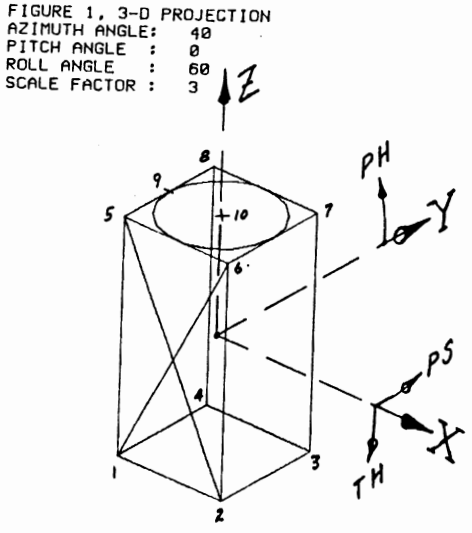
3-D Projections



by Bert Denaci

Have you ever wished to draw a three-dimensional object and then view it from any angle, including the plan, side, and end views? As an added bonus, would you like to see a perspective projection of the object in addition to the the usual orthogonal projection? The elementary Computer Aided Design (CAD) program *3d'view'object* demonstrates how easily this may be accomplished using COMAL plotting/drawing capabilities.

The figure below illustrates the orthogonal X-Y-Z axis system chosen, and a sample object (a prism with an X on one side and a circle on the top). The 3-dimensional coordinates of the object are defined in the initial position of the axes, XA-YA-ZA, as shown in the procedure coordinates. Eight points define the prism, and points 9 and 10, respectively, define a point on the circle and it's center. It is recommended that, as in this sample prism, the geometric center of the object be chosen as the origin of the coordinate system.



The axis system may then be displaced through the azimuth angle, PS, the pitch angle, TH, and the roll angle, PH; to the new positions XB-YB-ZB. Procedure transpose provides the trigonometric relations for converting the coordinates from the XA-YA-ZA axes to the XB-YB-ZB axes. Since the screen or a plotter can only *draw* on a single plane, we have chosen the XB-YB plane for the 3D projection, and the ZB dimension is used to define the perspective.

In a perspective drawing, the size of the object is inversely proportional to the distance from the observer. A perspective point, PP, value of 150 was chosen as the viewing distance from the origin of the coordinate system. As the value of PP increases, a perspective projection approximates an orthogonal projection. The choice between a perspective or orthogonal view is made by setting ZBIAS equal to either PP or 10000, respectively.

Procedure transpose'coordinates performs the necessary calculations for transposing the XA-YA-ZA to the XB-YB-ZB coordinates, and modifing the XB-YB values for the screen presentation. The term, PER(N), is the perspective correction for each coordinate, and M is the magnification factor. Also included is a horizontal screen correction factor, 1.35; and the offsets, 160 and 100 to center the object on the screen.

Procedure draw'object provides the statements to draw the projection of the 3-dimensional object on the XB-YB plane. It also provides for an 18 point calculation of the coordinates of the circle in the XA-YA plane for transformation to the XB-YB plane.

In addition to the procedures, the main

More ►

program must include a dimension (DIM) statement for the maximum number of items in the arrays, XA(), YA(), ZA(), etc. The desired viewing angles, PS, TH, PH; and scale factor, M; are the inputs to the program. These inputs are then printed at the bottom of the hi-res screen for viewing along with the resulting drawing.

By using the procedures, *proc.1520plotter* for COMAL 2.0 or *1520/drv.proc* for COMAL 0.14 from *Today disk #9*, the outputs can be directed to the Commodore 1520 Printer/Plotter. Programs *1520polygons.14* from *Today disk #9*, and *1520draw'house* from User Group disk #11, are examples of the conversion from screen to printer output. Programs, *3d-view'airplane*, *1520-3d'object*, and *1520-3d'airplane* on *Today disk #13* are additional applications of *3d'projection.* ■

Disk Talk

READ AND RUN

Our new **Read And Run** series of disks will contain COMAL programs as well as SEQ text files of documentation, instructions, or other program information. A master program lets you read the instructions or print them on your printer, two columns per page, just like a newsletter. Programs can be run directly or from a menu.

These disks provide us with a way of distributing more articles and programs. If you have a program you'd like to share, send us the program along with the "article" or documentation as a SEQ file on the disk. We will adapt it for this new system for you. There will be a version for both COMAL 0.14 and 2.0.

SHAREWARE - COMAL STYLE

Several people have sent us programs that they would like distributed by the Shareware system. This system is an alternate to commercially selling their programs. The programs are combined onto Shareware disks. These disks are free to be copied, included in User Group disk libraries, placed on BBS systems and passed out to friends.

A Shareware disk is a collection of programs that normally would have been sold commercially. Try the programs. Delete the ones you don't want. Keep the ones you like. Each program you keep has a message from the author requesting a donation from each user who keeps the program. Send your donation to the author at the address listed in the message.

At this time, programs include: COIN BBS, Proto-D Expert System, Traffic Calc, HazMat, and Finger Print Classifier. ■

Super Chip 

SUPER CHIP INFORMATION

Super Chip is a 16K EPROM designed to plug into the empty socket inside the black COMAL 2.0 cartridge. It works with both the C64 and C128, and features:

*** AUTO START SYSTEM. When the computer is turned on or reset, if the SHIFT key is depressed, the file named "hi" is loaded and run from disk.

*** FAST LOADER. Either *sizzle2.0* or the *rabbit* package will be included on Super Chip. Both load COMAL programs 2-5 times faster than normal. If *rabbit* is included, it will be disabled on power up. If you have a 1541 drive, enable it with these commands:

```
USE rabbit
setfast(TRUE)
```

These lines could be included in your "hi" program. Once fastmode is enabled, it remains that way until you specify otherwise. If *sizzle2.0* is included, we will try to have it automatically enable itself on power up.

*** C128 SUPPORT. Now COMAL 2.0, with Super Chip, can access most of the added features of the C128. See the article in this issue for details.

*** EXTRA COMMANDS. Super Chip adds about 100 new commands (in 9 packages) to the many already included in COMAL 2.0 and its 11 built in packages. These commands are listed in the Super Chip Command Chart in this issue.

Order a Super Chip at the same time as a COMAL 2.0 cartridge and we install and test it for you. Otherwise you may send us your cartridge and for an added $5 fee we will do the installation and testing.

AUTO START

The Auto Start part of Super Chip may be overlooked by many. Anytime the computer is reset (by turning the power on, pressing a reset button, or by a temporary power outage) the computer can automatically load and run the program on disk called "hi". Many people may get Super Chip just for this one feature! Some of the things it can do include:

* Set up the screen colors to your liking
* Define the function keys your way
* Present you a menu of programs to run
* Link in a user defined font
* Change your keyboard to Dvorak style
* Setup the default printer to match yours
* Setup your default disk drive unit
* Present a directory of the disk
* Run a BASIC program (via package BASIC)

We realize that many of you will not want COMAL to look for a "hi" program every time you turn on the computer. Therefore, it first checks the SHIFT key. If it is not depressed, the Auto Start is skipped. If SHIFT (or SHIFT LOCK) is depressed, the Auto Start will be attempted, after a short delay (allowing you to insert the disk **after** power on if you wish).

The "hi" program can be different on each disk. You could have the "hi" program present a menu of the program choices on the disk in the drive. It then could load and run the program you chose. This would be a very nice way to "package" a system of programs for others to use.

Notice: it seems that some C128 computers have a marginal power supply and may hang up during Auto Start. Pressing the reset button on the right side of the computer usually solves the problem. Apparantly new C128's come with a better power supply.

More ►

INSTALLING SUPER CHIP

WARNING: The EPROM chip can be damaged by static electricity. Hold it by the two short edges rather than long edges. Never pass a chip from one person to another. Before touching the chip, discharge any built up static electricity in yourself by touching a metal object.

1) Bring your cartridge and the chip (still in its anti-static holder) to your work area. You also will need a small phillips head screwdriver and optionally a small flat head screwdriver.

2) With the small phillips head screwdriver remove the screw from the cartridge back.

WARNING: During the next step, the cartridge may unexpectedly just pop open (and the printed circuit board could pop out). Keep the cartridge near the table or desktop and brace your arms.

3) Open the cartridge case using one of these methods:

3a) Place both thumbs in the open end of the cartridge. Pull **hard** to force the case apart.

3b) Release the four plastic tabs on the sides of the cartridge, one at a time. Insert flat headed screwdriver into a hole. Pull handle away from the cartridge, which causes the head to press toward the center of the cartridge. While doing this, pull apart the top and bottom at that point.

4) Set the cartridge on your table or desk top. The printed circuit board should be resting inside the bottom of the casing. The casing top can be set off to one side.

WARNING: Notice the two small ends of a ROM chip currently in the cartridge. One edge has a small half circle "notch" cut out. The other edge of the chip is flat. Super Chip also has a "notch" on one end. Super Chip must be inserted in the empty socket with its notch facing the same way as the other two chips. If inserted backwards, the chip will be ruined.

5) Make sure no static electricity is built up by touching a metal object.

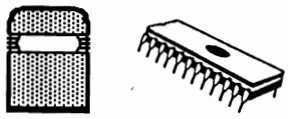
6) Carefully remove the Super Chip from its protective holder. Notice which end is the notched end. Insert the Super Chip into the empty socket, making sure that the notched end is facing the same way as the other two chips. If the pins seem a bit too wide for the holes, carefully hold the chip by the two ends sideways, and press each set of legs gently against a book cover, or something of that nature. This pushes each pin on the side slightly towards the middle. While inserting the pins on each side, be careful that ALL pins go into their respective holes in the socket. Watch that a pin doesn't bend in under the chip, or bend out over the socket. If a pin bends, you can straighten it back out, but then it is harder to insert, and it is possible to actually break the pin. If a pin breaks, the chip is ruined.

7) Once the chip is inserted, press down on it firmly to make sure it is positioned properly in the socket.

8) Replace the cover on top of the bottom of the case (with the board still in the bottom part). Press it firmly together (the four tabs will "snap" back together. The screw may be left out if desired (makes it easier to re-open the case later).

[We will install the chip and test it for a $5 added fee. Or if you order a cartridge and chip at the same time, installation and testing is free.] ■

C128 Package



by David Stidolph

If you own a C128, you will be interested in the package on *Today Disk #13*, called *pkg.c128*. Just LINK the package from disk and it automatically initializes the 80 column screen and numeric keypad. The package uses the cassette buffer and won't work with other programs which also use that buffer.

The 80 column screen is for output only (Super Chip includes an expanded version with proper inputs). You can print text to the 80 column screen by sending it to device 7 (the package directs device 7 output to the 80 column screen). Otherwise use the PRINT80 command (discussed later). To LIST the program in memory to the 80 column screen, use the following command:

LIST "u7:"

The 80 column screen automatically scrolls up a line when you print past the last column of the bottom line, or print a carriage return on the bottom line.

You can also use the SELECT OUTPUT or OPEN FILE commands to direct output to the 80 column screen ("u7:"). It is maintained by a chip called the Video Display Controller (VDC for short) which performs functions similar to the VIC chip in the C64.

The following is a description of the package commands. Proper syntax for each command can be found in the Super Chip Command Chart.

ATTRIBUTES

As with the 40 column screen, each character on the screen has its own color, but extra features are available to make your display more effective. The ATTRIBUTE command requires four parameters. Once the attributes are set, they will apply to all newly printed characters. Each parameter can be set with a 1, cleared with a zero, or ignored with a -1. The parameters, in order, are:

1) Alternate character set

This setting determines which character set is used for displaying the character. If on, the second (or upper) character definition is used. If off, the first (or lower) character set is used. This provides 512 possible characters for each character displayed.

2) Show character in reverse mode

This setting determines whether the character will be displayed in reverse mode. If set, any pixel in the character that normally is on will be displayed off, and any pixels normally off will be displayed on. This is different than PRINTing CHR$(18), which will print characters using the upper half of the current character set.

3) Underline the character

This setting determines whether or not the character will be underlined.

4) Blink the character

This setting determines whether or not the character is blinking.

EXAMPLES

```
attributes(1,0,0,0) //default
attributes(0,0,1,0) //underline only
attributes(-1,0,0,0) //skips alternate
```

More ►

BACKGROUND80, COLOR80

These two commands set the background color of the 80 column screen and the color of newly printed characters.

These commands take four parameters. In order they are: red, green, blue, and intensity (or bold). Each parameter can be set (1), cleared (0) or ignored (-1). The following tables show what color appears for the various settings.

Red	Green	Blue	Bold-off	Bold-on
0	0	0	Black	Dark Grey
0	0	1	Blue	Light Blue
0	1	0	Green	Light Green
0	1	1	Cyan	Light Cyan
1	0	0	Red	Light Red
1	0	1	Purple	Light Purple
1	1	0	Brown	Yellow
1	1	1	Light Grey	White

EXAMPLES

```
color80(1,1,1,1)      //white
background80(0,0,0,0) //black
color80(-1,-1,-1,0)   //clear intensity
color80(1,-1,-1,-1)   //add red to color
```

DISPLAY80

This command allows you to change the 80 column screen to show a variable number of lines on the screen. The VDC has 16K of its own memory (half of which is used for the character set definitions). The C128 package allows you to set the number of lines displayed on the screen (1 to the maximum your monitor can show) and which line the screen should start with.

Think of the 80 column display being a window that can slide up and down. For example, *display80(1,25)* gives the default 25 lines starting at the top of memory. You could set the window to start at the second line with the command *display80(2,25)*. To get 10 lines displayed from the current top line, use *display(-1,10)*. Memory limits the total number of lines for your sliding widow to 50 lines. If you exceed this limit, you will overwrite attribute memory. If you need more lines, you can turn off the attributes with *monocolor80(TRUE)*. Most monitors can display 27 or 28 lines.

HARDCOPY80

This command has one parameter, a filename (specifying the output location). The text on the 80 column screen is printed to the file you specify, normally "lp:" to go to the printer. It also can send the text to a disk file, the 40 column screen ("ds:"), or to the modem ("sp:").

PAGE80

This command clears the current display, no matter how big or small, without affecting the area outside the display. It also moves the cursor to the top left hand corner, and clears all attributes if monocolor is not in effect.

MONOCOLOR80

This command enables or disables the attributes. It requires a single parameter, which should be either TRUE or FALSE. If TRUE, monocolor mode will be in effect - disabling attributes, and doubling text space (100 lines). In monocolor mode , COLOR80 changes the color of all characters, not just the newly printed ones.

More ►

SWAPFONT80

Since monocolor mode does not use attributes, all characters are taken from the first definition (uppercase/graphics). SWAPFONT80 copies the (lowercase/ uppercase) definition there. To recover the first definition, use INIT80.

CURSOR80

This command determines where the cursor is on the 80 column screen. The first parameter is the row number and the second parameter is the column number. Numbering is the same as with the normal CURSOR command. A value of zero is used to leave the present row or column position alone. A value of 1 indicates the leftmost position for row, and the top line for column. Positioning is relative, and limited to the current DISPLAY80 command. If you attempt to use this command to go outside of the current display, a *value out of range* error (#5) will be given.

PRINT80

This command provides output to the 80 column screen without opening a file, or selecting output to device 7. It prints a string to the 80 column screen at the location specified by first two parameters. If a carriage return is desired, it must be included within the text.

CURCOL80, CURROW80

These two functions return the location where the next character will be printed. The numbering for the cursor position is explained under CURSOR80.

TURBO

This command can turn on the double speed mode of the C128. In this mode the 40 column screen cannot be displayed, so it is turned off (the 80 column screen remains on if activated). Any disk access with the turbo speed activated will cause errors, and may lock up your computer.

KEYPAD

By default the extra keys on the C128 are enabled. Since the keys are scanned 60 times every second, this extra step can slightly slow your program. For timing tests I suggest you use **keypad(FALSE)**.

VDC REGISTERS

The following commands allow you to examine and alter the VDC registers.

SET80

This command has two parameters. The first is a register number (0-36). The second is the number to place in it (0-255).

READ80

This function returns the value in the specified VDC register.

INIT80

This command restores the registers to the values they recieve on power up, clears VDC memory, and writes both character sets to VDC memory.

[The disk based pkg.c128 package is a non-rommed package. This means it can be linked onto your programs. Then, when your program is saved, the package is saved with it. It also can be discarded.] ■

VDC Editor

by David Stidolph

Vdc'editor on *Today Disk #13* uses the C128 package to put a sample display on the 80 column screen, and a sliding bar menu on the 40 column screen - both at the same time. The package is already linked to the program. All you need to do is issue the command:

run"vdc'editor"

The program will only run on a C128. You need a monitor that can be switched between the 40 and 80 column displays (or two separate monitors) to use this program.

The program allows you to play with the VDC registers by name, not number. Don't worry if you don't understand a name. Use the cursor keys to select the VDC function (from the 40 column screen) you wish to change. The plus (+) and minus (-) keys increment and decrement register values, immediately changing the 80 column screen. Every feature of the VDC can be tested. Write down interesting register values for use in your programs (CTRL P gives a hardcopy of the screen). Using this program and a Zenith RGB monitor, I was able to display 30 rows of 93 columns.

Try turning on both **Video Mode** and **Interlace Mode**, increasing **Lines Per Char** to 11 and **Vert Fine Adjust** to 6. If you increase the number of displayed lines (**Vert Displayed**), also increase **Vertical Sync Position** to move the screen upwards. Remember to switch back and forth between the 40 and 80 column displays.

For more information on what these functions do and how to manipulate them, refer to the *Commodore 128 Programmer's Reference Guide.* ■

Package Version

Each package in Super Chip has a version command (version'name$). This is a deliberate attempt to set a standard, letting users know if they have an old or updated version of a package. For example, to find out which version of the *math* package you have:

PRINT version'math$

Replace math with name of the package in question. It is **required** that the first 5 characters of the string represent the numeric version of the package. A program can test for current version like this:

if val(version'c128$(1:5)) >= 1.1 then

This tests to see if you have version **1.1** or later of the **c128** package. Other information may also be included in the version string such as the name of the package and programmer, or comments about the package.

If you intend to program your own packages, please include a version command. The following code can be adapted into your source code:

```
verstr lda #verst3-verst2+2
       jsr excgst
       ldy #00
verst1 lda verst2,y
       sta (copy2),y
       iny
       cpy #verst3-verst2
       bne verst1
       lda #00
       sta (copy2),y
       lda #verst3-verst2
       iny
       sta (copy2),y
       rts
verst2 .byt ' 1.02 strings package'
       .byt ' by Richard Bain'
       .byt 13
       .byt '      quicksort code '
       .byt ' by Robert Ross'
       .byt 13
       .byt '      string code    '
       .byt ' by David Stidolph'
verst3
```

■

C128 Package --- Extra Commands

by David Stidolph

The c128 package on Super Chip has everything described in the C128 package article, as well as additional commands. Programs written using the LINKable C128 package will work unchanged with Super Chip. However, programs written using these extra Super Chip commands will require Super Chip to work:

CLEAR80

This command is used to clear all the VDC memory for 80 column graphics.

GETCHAR80$

This string function returns the 16 byte definition of a character in VDC memory.

INKEY80$

This string function works just like INKEY$. A blinking cursor is provided while the program waits for a key to be pressed. That key is returned.

INPUT80$

This string function provides a protected input field on the 80 column screen. The first two parameters specify the starting position and the third specifies the maximum input length (0-252).

LINE80 & PLOT80

LINE80 draws a line connecting two x-y points on the 80 column graphics screen; PLOT80 plots an x,y point.

SETCHAR80

This command defines a VDC font character pattern specified by the string parameter. If the string is less than 16 bytes in length, zeros are added to make a proper VDC character definition.

SETGRAPHIC80

This command initializes the VDC for 640 by 200 graphics. The entire VDC memory is used as a gigantic bit-map display, addressed by conventional x-y graphics (x ranges 0-639 and y ranges 0-199). Location 0,0 is in the lower left hand corner. If you attempt to access a point outside of these bounds, a *value out of range* error will result. The TURBO command will double the graphics speed.

SETPEN80

This command sets how PLOT80 and LINE80 work. If you pass this command a 1, then points will be set. If you pass this command a 0, then points will be cleared. If you pass a -1, then points will be inverted. This last setting is useful for drawing the same line twice - the second time returns all points to the state they were before the first line was drawn.

SETTEXT80

This command executes INIT80 and sets the VDC to display text. Any text or graphics previously on the 80 column screen is lost. ■

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

CLASSIFIED AD

Beige COMAL 2.0 cartridge for sale. Low mileage. $45 asking price. William Klopfer, Code 4023 Navairdevcen, Warminster, PA 18974. ■

Super Chip Commands

C128

Initialize with: **USE C128**
The 80 column screen is treated as device 7.

SELECT OUTPUT "U7:" *//80 column screen*

ATTRIBUTES-sets text screen attributes
attributes(<alt>,<rvs>,<underline>,<blink>)
attributes(0,0,1,1) //underline blinking

BACKGROUND80-sets background color
background80(<red>,<green>,<blue>,<bold>)
background80(0,0,0,0) // black

CLEAR80-zeros the VDC RAM for graphics
clear80
clear80 // for 80 col graphics

COLOR80-sets current color
color80(<red>,<green>,<blue>,<bold>)
color80(1,0,0,1) // light red

CURCOL80-returns current cursor column
curcol80
x=curcol80

CURROW80-returns current cursor row
currow80
y=currow80

CURSOR80-positions the cursor
cursor80(<row>,<column>)
cursor80(5,1)

DISPLAY80-sets display window area
display80(<topline>,<number of lines>)
display80(5,15) // middle 15 lines

GETCHAR80$-returns 16 byte definition
getchar80$(<font# 0-1>,<char# 0-255>)
a$:=getchar80$(0,1) // an "A"

HARDCOPY80-80 col text screen output
hardcopy80(<output loc$>)
hardcopy80("lp:")

INIT80-completely initialize VDC for text
init80
init80

INKEY80$-waits for key to be pressed
inkey80$
reply$=inkey80$

INPUT80$-input string on 80 col screen
input80$(<row>,<col>,<length (0-252)>)
reply$=input80$(10,5,1)

KEYPAD-enables / disables extra keys
keypad(<on/off>)
keypad(false) // turn off keypad

LINE80-draws a line on 80 column screen
line80(<start x>,<start y>,<end x>,<end y>)
line80(0,0,50,50)

MONOCOLOR80-no attribute memory
monocolor80(<on/off>)
monocolor80(true)

PAGE80-clears 80 col screen display area
page80
page80

PLOT80-plot, unplot, or flip pixel
plot80(<x>,<y>) // see setpen80
plot80(x,y)

PRINT80-prints text on 80 column screen
print80(<row>,<col>,<text$>)
print80(24,1,"Press Return")

READ80-return the value in VDC register
read80(<register#>)
display'lines:=read80(6)

SET80-set a value into VDC register
set80(<register#>,<value>)
set80(29,3) // underline now is strikeout

More ►

SETCHAR80-defines a font character
setchar80(<font 0-1>,<char 0-255>,<pattern$>)
setchar80(0,1,newchar$) // change "a" in font

SETGRAPHIC80-sets VDC into graphics
setgraphic80
setgraphic80

SETPEN80-defines what plot80/line80 do
setpen80(<type>)// -1=flip, 0=erase, 1=draw
setpen80(1)

SETTEXT80-sets VDC into text mode
settext80
settext80

SWAPFONT80-lowercase monocolor font
swapfont80
swapfont80

TURBO-set fast mode in computer on/off
turbo(<true/false>)
turbo(true)

VERSION'C128$-version, author
version'c128$
PRINT version'c128$

MATH

Initialize with: **USE MATH**

BIN$-returns a number in binary form
bin$(<number (0-255)>)
PRINT bin$(255)

DISTANCE-the distance between 2 points
distance(<x1>,<y1>,<x2>,<y2>)
length:=distance(a,b,x,y)

EVEN-returns true if number is even
even(<number (-32768 - 32767)>)
IF even(num) THEN

GCD-returns greatest common divisor
gcd(<positive integer>,<positive integer>)
PRINT gcd(num1,num2)

GCF-returns greatest common factor
gcf(<positive integer>,<positive integer>)
PRINT gcf(num1,num2)

HEX$-returns a number in hex form
hex$(<number (0-65535)>)
PRINT hex$(255)

HYPOTENUSE-long side of right triangle
hypotenuse(<number>,<number>)
hypotenuse(3,4)

LCM-returns least common multiple
lcm(<positive integer>,<positive integer>)
PRINT lcm(num1,num2)

MAX-returns the larger of 2 numbers
max(<number1>,<number2>)
high:=max(n,high)

MAXINT-returns largest integer allowed
maxint
IF number>maxint THEN

MIN-returns the smaller of 2 numbers
min(<number1>,<number2>)
low:=min(n,low)

ODD-returns true if the number is odd
odd(<number>)
IF odd(num) THEN

PRIME-returns true if number is a prime
prime(<number (0-65535)>)
IF prime(num) THEN PRINT num;

ROUND-rounds number to nearest integer
round(<number>)
score:=round(score)

TRUNC-removes fractional part of number
trunc(<number>)
number:=trunc(number)

VERSION'MATH$-version, author
version'math$
PRINT version'math$(6:);version'math$(1:5)

More ►

FILES
Initialize with: **USE FILES**

BLOAD-load bytes from disk to memory
bload(<filename$>)
bload(name$)

BSAVE-save memory to disk
bsave(<filename$>,<mem start>,<mem end>)
bsave("testfile",$c000,$cfff)

COLLECT-validates disk in current drive
collect
collect

COUNTSP-serial port buffer char count
countsp
IF countsp>5 THEN

CURRENT'FILE$-last accessed filename
current'file$
PRINT "File was:";current'file$

DRIVE8-changes device 9 to be device 8
drive8
drive8

DRIVE9-changes device 8 to be device 9
drive9
drive9

FILE'EXISTS-returns true if file on disk
file'exists(<filename$>)
IF file'exists(name$) THEN

GETSP$-returns string from serial port
getsp$(<file#>,<count (0-256)>)
reply$=getsp$(2,0)

SAVEIT-saves program to current filename
saveit
saveit

TYPE-prints ASCII file contents
type(<filename$>)
type("program.doc")

VERSION'FILES$-version, author
version'files$
IF VAL(version'files$(1:5))<2 THEN

KEYBOARD
Initialize with: **USE KEYBOARD**

CLEARKEYS-clear keyboard buffer
clearkeys
clearkeys

DISABLE'KEYBOARD-all keys disabled
disable'keyboard
disable'keyboard

ENABLE'KEYBOARD-all keys enabled
enable'keyboard
enable'keyboard

FILLKEYS-put text$ into keyboard buffer
fillkeys(<text$>) // up to 10 characters
fillkeys("yes"+chr$(13))

LOCK'UPPERCASE-upper/graphic char
lock'uppercase
lock'uppercase

LOCK'LOWERCASE-upper/lower case
lock'lowercase
lock'lowercase

REPEATKEYS-keys repeat if held down
repeatkeys(<on/off>)
repeatkeys(true)

SHIFT-returns true if shift key depressed
shift
WHILE NOT shift DO

UNLOCK'CASE-enable shift/commodore
unlock'case
unlock'case

VERSION'KEYBOARD$-ver/author
version'keyboard$
PRINT version'keyboard$(6:)

More ►

COLORS
Initialize with: **USE COLORS**

The following color names may be used:

0 = black, grey0
1 = white, grey4
2 = red
3 = cyan
4 = purple
5 = green
6 = blue
7 = yellow
8 = orange
9 = brown
10 = light'red, lt'red, lred, pink
11 = dark'grey, dgrey, grey1
12 = medium'grey, mgrey, grey2, grey
13 = light'green, lt'green, lgreen
14 = light'blue, lt'blue, lblue
15 = light'grey, lt'grey, lgrey, grey3

Examples:
background(black)
pencolor(grey)
border(white)

Plus the string function:

VERSION'COLORS$-version, author
version'colors$
PRINT version'colors$

SYSTEM2
Initialize with: **USE SYSTEM2**

CURRENT'PAGE-peek/poke memory page
current'page
CASE current'page OF

HIDEAWAY-hide program lines
hideaway(<end'line>)
hideaway(800)

HIDESCREEN-blank 40 col screen
hidescreen
hidescreen

HOST$-name of computer (c64 or c128)
host$
IF host$:="c128" THEN

PAUSE-pause for specified seconds
pause(<number of seconds>)
pause(8.3)

REVEAL-restore non-listable lines
reveal
reveal // all lines now listable

SCROLL'DOWN-all screen lines down
scroll'down
scroll'down

SHOWSCREEN- 40 col screen visible
showscreen
showscreen

VERSION'SYSTEM2$-version, author
version'system2$
ver=VAL(version'system2$(1:5))

STRINGS
Initialize with: **USE STRINGS**

FLOAT$- 5 byte internal floating point
float$(<number>)
this'one$:=float$(3.5)

LOWERCASE$-convert all letters to lower
lowercase$(<text$>)
PRINT lowercase$(reply$)

QUICKSORT-sort string array
sort(<array$()>,<integer>,<integer>)
sort(name$(),1,lastitem) //ascending
sort(name$(),lastitem,1) //descending

STRING$-returns repeated string
string$(<string$>,<count>)
PRINT string$("warning ",10)

SWAP'STRING-exchange two strings
swap'string(<text$>,<text$>)
swap'string(friend$,enemy$)

More ►

SWAP'INTEGER-exchange two integers
swap'integer(<integer1>,<integer2>)
swap'integer(myscore#,yourscore#)

SWAP'REAL-exchange two real numbers
swap'real(<number1>,<number2>)
swap'real(income,taxes)

UPPERCASE$-convert all letters to upper
uppercase$(<text$>)
reply$:= uppercase$(reply$)

VERSION'STRINGS$-version, authors
version'strings$
author$:= version'strings$(6:)

AUTOSTART

Initialize with: **USE AUTOSTART**

Provides a system reset that will restart the Auto Start system on Super Chip.

RABBIT

may be replaced by SIZZLE2.0

Initialize with: **USE RABBIT**

BLOCKMODE-TRUE if blockmode enabled
blockmode
IF blockmode THEN

BREAD-read a block
bread(<track>,<sector>,<buffer>,<text$>)
bread(18,0,1,c$)

BWRITE-write a block to disk
bwrite(<track>,<sector>,<buffer>,<text$>)
bwrite(18,0,1,c$)

DEVICE-returns current device number
device
this'one=device

FAST-returns status of fastmode
fast
IF fast THEN

OFFTEST-returns status of SETOFFTEST
offtest
if offtest then

RDERR-returns last disk error number
rderr
IF rderr THEN

SECTOR-returns number of next sector
sector
next'one:=sector

SETBMODE-set fast block mode
setbmode(<on/off>)
setbmode(TRUE)

SETDEVICE-set drive number
setdevice(<number>)
setdevice(other'drive)

SETFAST-set fast mode
setfast(<on/off>)
setfast(TRUE)

SETOFFTEST-controls device testing
setofftest(<integer>)
setofftest(TRUE)

TRACK-returns number of next track
track
next'track:=track

VERSION'RABBIT$-version, author
version'rabbit$
PRINT version'rabbit$ ■

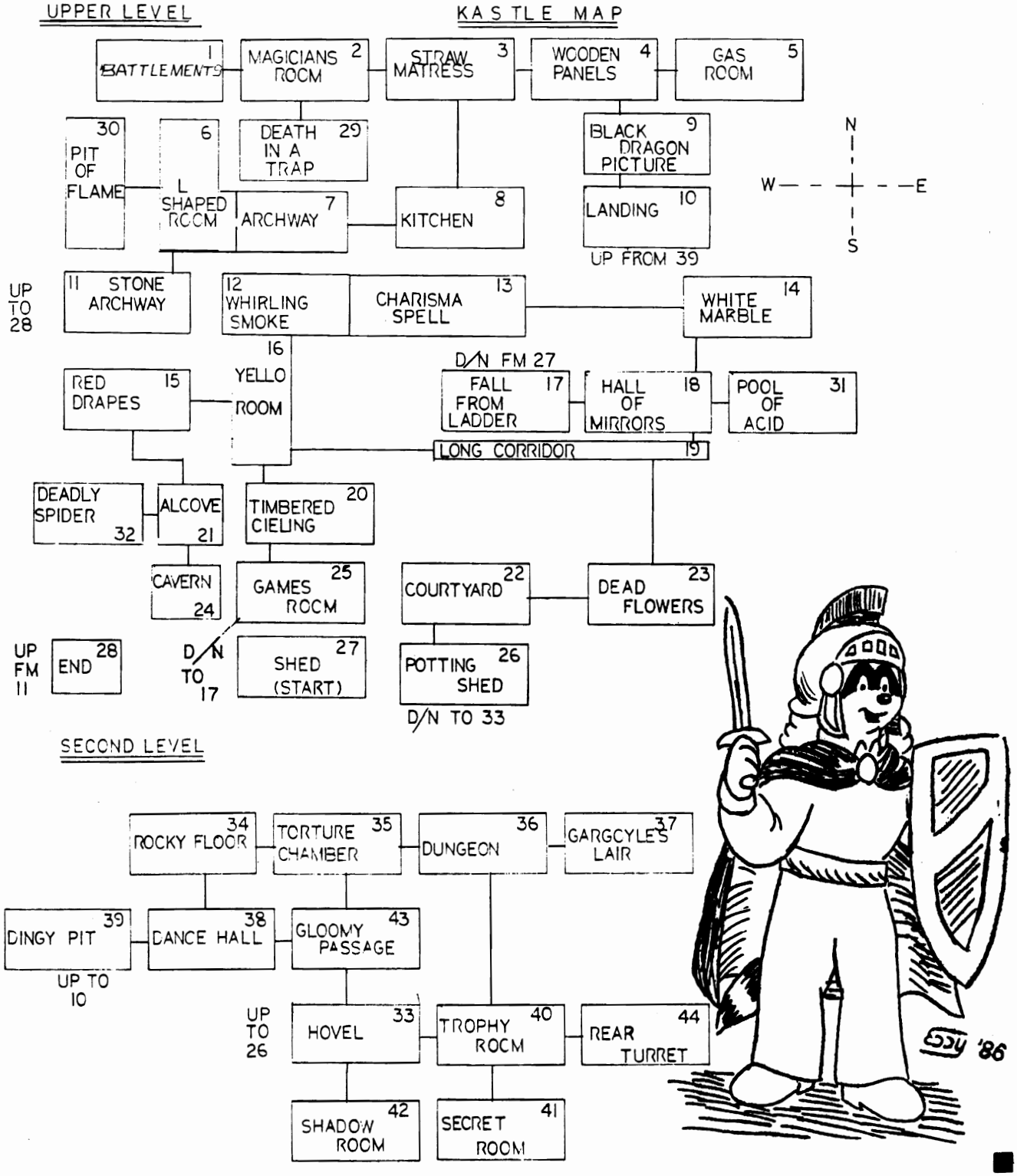
SUPER CHIP INSTALLED?

```
FUNC superchip CLOSED
 USE system
 setpage($84); chip:=TRUE
 RESTORE
 FOR x:=$8014 TO $8018 DO
  READ num
  IF PEEK(x)<>num THEN chip:=FALSE
 ENDFOR x
 RETURN chip
 DATA 4,67,49,50,56
ENDFUNC superchip
```

■

Kastle

An adventure game on *Today Disk #13* by Richard Aurland.



Prime Factorization

Help From Super Chip

by Steve Kortendick

One of the basics of any work in arithmetic is the use of prime numbers and the prime factorization of composite (non-prime) integers. Such factorizations are so central that proof of their unique existence is usually called **The Fundamental Theorem of Arithmetic**; that every integer greater than one can be written as the product of its prime factors in one way, unique except for the order of the primes. That is, every number has a unique prime factorization.

You have probably struggled to factor numbers on occasion, or perhaps you have invested in one of the books of tables which list these factorizations. Rohnius published just such a table in 1659, but even if you have it, the following routine can still be of significant help; Rohnius' table only went through 24000 but the following routine will give you prime factorizations for numbers through 65535. Of course if you have Felkel's 1776 book, you'll have all the factors through 408000, but most of those books were used for their paper in order to make artillery shells in the Austrian war against the Turks. If you have that book, you can sell it to a collector for a small fortune and still use this routine. (Tidbits are from Beck, Bleicher, Crowe: Excursions into Mathematics, Worth Publishers, New York, 1969, pp. 102-3.)

Two functions of the math package in the Super Chip, **prime** and **gcf**, make this routine simple. Those, coupled with COMAL's simple recursion, made the routine a dream to write. I considered a version in BASIC just for comparision, but soon realized the mess I'd get into.

The REPEAT loop is the key to this code. Starting with 1, continue adding 1 to candidates for prime factors until you find one. **Prime(trial)** says if the number is prime. **Gcf(number,trial)=trial** recognizes the fact that trial's only factors are itself and 1 (it passed the prime test). If **trial** is a factor of **number**, it is the greatest common factor. That one line, then, continues trying potential factors until it finds a prime factor. Of course by dividing the prime factor out, the rest of the factors are the same as those of **number/trial**. The recursion ends when a prime factor is all that's left.

To illustrate, let's use 75. We REPEAT, trying possible prime factors starting at 2. 3 works (prime(3) and gcf(75,3)=3 are both true) so we print it. Then go back for prime factors of 75/3, or 25. So we again start trying numbers 2 through 5 until 5 works as a prime divisor of 25. Then recurse with 25/5, or 5. Since that's prime, we print it and quit.

One would think that since we know (in the above example) that when we get a factor of 75, that we need not start with candidates as low as 2 to factor 75/3 or 25; we can start where we left off in the recursive steps. I tried passing a second parameter to the routine, the starting candidate. That way, in factoring 25 we start with 3 rather than 2 (or more significantly, in factoring 4913 we start with 17 the second time, rather than 2). However, the extra parameter passing actually slowed performance of large numbers, those for which it should be more significant. I suspect it's due to stack activity. This also shows, of course, that **prime** and **gcf** are both very fast routines.

Clearly the first few lines of code simply call the routine for the numbers 2 through 100, to be adapted as necessary. The heart of the matter is the **print'factors**

More ►

procedure. Naturally, you may want to do more than just print the factorizations. Simply adapt the routine to stuff them into an array or whatever you need to do for your application.

This is just a small example of a practical application of the power of COMAL (especially in the REPEAT and recursive structures) coupled with two new very handy functions in the Super Chip's math package. Sure makes life easier.

```
FOR number:=2 TO 100 DO
  PRINT USING "###### = ": number;
  print'factors(number)
  PRINT
ENDFOR number
//
PROC print'factors(number) CLOSED
  USE math
  IF prime(number) THEN
    PRINT number;
  ELSE
    trial:=1
    REPEAT trial:+1 UNTIL prime(trial) AN
    D THEN gcf(number,trial)=trial//wrap
    PRINT trial;
    print'factors(number/trial)
  ENDIF // prime(trial)
ENDPROC print'factors
```

VOCABULARY 0.14

by Henry Farkas

A series of programs on *Today Disk #13* are set up to help kids study vocabulary or spelling. I did this in several programs rather than in one because memory is limited in COMAL 0.14. Use of the program is pretty self evident once the main menu is showing. If there's interest, I can easily transfer the program to COMAL 2.0.

Questions

SPLITSCREEN & SAVESCREEN

Question: When using COMAL 2.0, the SPLITSCREEN command does not seem to work while in multi-color graphics mode. Also, the SAVESCREEN command then creates a 40 block file rather than the 36 block file created while in hi-res graphics mode. Why? - Fred Staudaher, Alexandria, VA

Answer: SPLITSCREEN is only available while in hi-res mode. The extra blocks in the file created by SAVESCREEN, while in multi-color mode, are for the colors of each pixel. This is how to switch to each of the two types of graphics screens in COMAL 2.0:

```
USE graphics
graphicscreen(1) // multi-color
graphicscreen(0) // hi-res
```

CHECKS HOLD

Question: I noticed that the wait for a check to clear the bank delays the shipment of an order for about 2 weeks. Is it possible to send you a check for $50 or $100 now, and have it credited to my subscriber account? Then I could order items later using the credit in the account and not have the added delay. - Richard Aurland, Las Vegas, NV

Answer: Yes. A few COMALites are already doing this. Our order processing system (a huge COMAL program) remembers any credit or amount due for every subscriber. Once you are a subscriber, you can "overpay" by as much as you want, and our system will remember it. However, with each order, you should clearly state that there is a credit in your account for the order. Otherwise, we might send it back thinking you forgot to enclose your check.

Bug Fixes

NO BUG IN WORD GAME

I beg to differ with Peter Gilbert (*Bug Fix, COMAL Today #11, page 8*), but there is no bug in option 5 of *Word Game* (*COMAL Today #8, page 69*). I just tried out the disk you sent me, the *official version* as you edited it, and option 5 works just fine with the file *comal*, the demonstration which went along with the program.

It may be that in entering his own files, Gilbert failed to read the directions, which stated to leave an empty space at the beginning **and** at the end of the text being entered. Not leaving a space at the end may have caused the problem which his *fix* fixes, but if the directions for text entry are followed, there should be no problem. Not that the program is perfect; not that I don't appreciate useful suggestions; but option 5 is fine. - Danial Horowitz

TODAY DISK #12 NOTE

The *Font Editor* on the front side of *Today Disk #12* is a very large program. Even before it is run, the program extends into the expanded memory area of COMAL 0.14. The *HI* program does properly expand the memory, but it seems that a NEW command must be issued prior to trying to LOAD or CHAIN the *Font Editor* program. To run the *Font Editor* do this:

1) Quit the HI program
2) Type: **new** *<return>*
3) Type: **chain "hi"** *<return>*
or Type: **chain "font'editor"** *<return>*

On the 2.0 side, the HI program provides a menu of 3 programs, including *wheel of fortune*. Unfortunately, the *wheel of fortune* program was delayed, and is on *Today Disk #13* instead. Do not choose that program from the menu.

MISSING IRQ'PROC

Richard Mayor pointed out an error in *Using the Interrupt Command, COMAL Today #8*, page 62-63. The demo program on page 63 should add the following line at the very start of the program:

INTERRUPT irq'proc

TRANSFER FIX

Null parentheses need to be added on page 45 of *COMAL Today #12* in the article *Transfer 0.14 Programs To 2.0*:

COMAL 0.14:

FROM: proc show'it(ref player$,table)
TO: proc show'it(ref player$(),table())

FFDB FIX

An example data file is used and referred to by the *FFDB* (*Free Form Data Base*) on pages 40-41 of *COMAL Today #12*. A data file name is prefixed with **DAT.** in COMAL 2.0, while it is suffixed with **.DAT** in COMAL 0.14. Since the *FFDB* program works with both 0.14 and 2.0, we debated which convention to use. After deciding to go with DAT. we forgot to change the file name on page 40. The correct data file name is *DAT.FFDB* and each place that shows it as *FFDB.DAT* should be changed to *DAT.FFDB*.

ROADS TO ROME 0.14 FIX

A small fix is needed for the examples throughout the *Roads To Rome* article in *COMAL Today #12*, pages 21-23. To work in COMAL 0.14, each line that increments the string **res$** must be changed (six places):

FROM: **res$:+<text>**
TO: **res$:=res$+<text>** ■

Today Disk #12 - Front

			126 Files	0 Blocks Free:
boot c64 comal+	qlinksimulator	~character sets~	payment.proc	decimal.func
enhancer-1541	sprite'designer2	~~~~~~~~~~~~~~~~	quote'mode.proc	basic-sieve
enhancer-others	~~~~~~~~~~~~~~~~	font.runes	showkeys.proc	basic-trig
c64 comal 0.14	~ warning ~	set.art deco.b	singlesided.proc	~~~~~~~~~~~~~~~~
comalerrors	~ all files ~	set.standard.b	slow.proc	~ pascal ~
ml.sizzle	~ below this ~	~~~~~~~~~~~~~~~~	str.proc	~~~~~~~~~~~~~~~~
sizzle.14e	~ point cannot ~	~ functions ~	~~~~~~~~~~~~~~~~	pascal-ahls
hi	~ be loaded ~	~~~~~~~~~~~~~~~~	~ benchmark ~	pascal-misshd3
menu	~~~~~~~~~~~~~~~~	enhanced.func	~ programs ~	pascal-sieve
menu-enhancer	~ data files ~	keyaddress.func	~~~~~~~~~~~~~~~~	pascal-trig
programs.dat	~~~~~~~~~~~~~~~~	val.func	~ promal ~	~~~~~~~~~~~~~~~~
programs-e.dat	beachnames.dat	~~~~~~~~~~~~~~~~	~ programs on ~	~for more info ~
comalhelp.txt	beachsprites.dat	~ procedures ~	~ other side ~	~ send sase to ~
~~	beachverbs.dat	~~~~~~~~~~~~~~~~	~ of disk ~	~~~~~~~~~~~~~~~~
~comal programs~	dat.6510opcodes	back'side.proc	~~~~~~~~~~~~~~~~	~ comal users ~
~~	dat.ffdb	bell.proc	~ comal ~	~ group, usa ~
build'beach'file	doubler.doc	convert1.proc	~~~~~~~~~~~~~~~~	~ 6041 monona ~
check'book	objecttable.dat	convert2.proc	comal-ahls.l	~ madison, wi ~
create'beach	verbtable.dat	convert3.proc	comal-misshd3.l	~ 53716 ~
double'column	~~~~~~~~~~~~~~~~	defkey.proc	comal-sieve.l	~~~~~~~~~~~~~~~~
font'editor	~listed program~	doublesided.proc	comal-trig.l	~(608)222-4432 ~
free'form'db	~~~~~~~~~~~~~~~~	errtext.proc	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~
kellys beach	~ use enter ~	fast.proc	~ basic ~	
load'font/demo	~~~~~~~~~~~~~~~~	front'side.proc	~~~~~~~~~~~~~~~~	
monitor	roads'main.l	joystick.proc	basic-ahls	
pic finder	~~~~~~~~~~~~~~~~	paddle.proc	basic-misshd3	

Today Disk #12 - Back

			127 Files	4 Blocks Free:
hi	~~~~~~~~~~~~~~~~	proc.front'side	~ video basic ~	~~~~~~~~~~~~~~~~
~~	dat.6510opcodes	proc.italics	~~~~~~~~~~~~~~~~	lightbulb.crg
~comal programs~	dat.ffdb	proc.payment	eclipse.graph	~~~~~~~~~~~~~~~~
~~	sawtooth.dat	proc.read'dir	~~~~~~~~~~~~~~~~	~ benchmarks ~
3d'fractals	sawtooth.dat'num	proc.singlesided	~flexidraw 5.0 ~	~~~~~~~~~~~~~~~~
adjust circle	square.dat	proc.slow	~~~~~~~~~~~~~~~~	~ promal ~
createdata'stmt	square.dat'num	~~~~~~~~~~~~~~~~	fdpirate	~~~~~~~~~~~~~~~~
check'book	~~~~~~~~~~~~~~~~	~picture files ~	~~~~~~~~~~~~~~~~	promal-ahls.s
draw'planet	~ packages ~	~~~~~~~~~~~~~~~~	~ koala ~	promal-misshd3.s
free'form'db	~~~~~~~~~~~~~~~~	~comal pictures~	~~~~~~~~~~~~~~~~	promal-sieve.s
fourier	pkg.oki92	~~~~~~~~~~~~~~~~	light house	promal-trig.s
monitor	pkg.oki92-rommed	hrg.sawtooth	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~
oki92-colortest	pkg.rabbit	hrg.square	~blazing paddle~	~for more info ~
pic finder	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ send sase to ~
read'dir/demo	~ procedures ~	~ basic bitmap ~	pi.colorwatch	~~~~~~~~~~~~~~~~
show&stamp	~ and ~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ comal users ~
sideways60	~ functions ~	a basic bitmap	~ paint magic ~	~ group, usa ~
sideways80	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ 6041 monona ~
~~	func.decimal	~ doodle ~	two girls	~ madison, wi ~
~listed program~	proc.back'side	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~ 53716 ~
~~	proc.base'conv	baseball cover	~printshop icon~	~~~~~~~~~~~~~~~~
~ use enter ~	proc.convert1	~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~	~(608)222-4505 ~
~~	proc.convert2	~ scribbler ~	c-64	~~~~~~~~~~~~~~~~
lst.roads'main	proc.convert3	~~~~~~~~~~~~~~~~	disk drive	
~~	proc.doublesided	scr.scribbler	~~~~~~~~~~~~~~~~	
~ data files ~	proc.fast	~~~~~~~~~~~~~~~~	~ compact pix ~	

Today Disk #12 Contributors

Phil Bacon

Phyrne Bacon

Paul Baker

Marcel Bokhorst

Ed Bolton

Ray Carter

Captain COMAL

Glen Colbert

Bert Denaci

Brian Fowler

Brian Grainger

Dick Klingens

Len Lindsay

Ed Mathews

John McCoy

Ralph McMullen

Kevin Quiggle

Joel Rea

Terry Ricketts

Patrick Roye

D. Shellenberg

David Skinner

David Stidolph

Colin Thompson

Jim Ventola

Joe Visser

Lowell Zabel

HOW TO TYPE IN PROGRAMS

Line numbers are required for your benefit in editing a program (but are irrelevant to a running program). Thus line numbers often are omitted when listing a COMAL program. It is up to YOU to provide the line numbers. Of course, COMAL can do it for you. Follow these steps to enter a COMAL program:

1) Enter command: NEW
2) Enter command: AUTO
3) Type in the program
4) When done: Version 0.14: Hit <return> key twice
   Version 2.0: Hit <STOP> key

While entering a program, use unshifted letters. If letters are capitalized in the listing it does not mean to use SHIFT with those letters. They are capitalized merely to be easy to read. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You DO have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add //wrap at the end. You must type it as one continuous line. Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789 '] [<backslash> _

The <left arrow> key in the upper left corner of the keyboard are valid. COMAL 2.0 converts it into an underline. If you see an underline in a program listing, type the <left arrow> key. The C64 and C128 computers use a <british pound> in place of the <backslash>.

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER:_____ July 1986
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description (all disks Commodore 1541 format) Prices subject to change

BOOKS:

(Canada & APO / FPO shipping add $1 more per book)

[ ] _____ **$6.95/$4.95 COMAL Today--The INDEX, Kevin Quiggle (est 64 pages-due August 86)(ship $1)**
[ ] _____ **$14.95/$9.75/$7.95 The INDEX on disk! Let your computer find the article/author! (due Aug 86)**
(COMAL Today issues 1-12, articles, notes, authors -- indexed! Buy book; add $7.95 for disk)
[ ] _____ **$19.95/$17.95 Introduction to Computer Programming, J William Leary (272 pages)-(ship add $3)**
[ ] _____ **$6.95/$4.95 Answer Book to Introduction to Computer Programming (64 pages)-(ship add $1)**
(Beginners text book for American students, spiral bound for easy use)
[ ] _____ $18.95/$16.95 COMAL Handbook, 2nd Edition, Len Lindsay (479 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Detailed Reference book to COMAL 0.14 and 2.0)
[ ] _____ $17.95/$15.95 Starting With COMAL, Ingvar Gratte (212 pages)-(shipping add $3)
(Beginners text book, Jr/Sr High level, by Swedish professor)
[ ] _____ $19.95/$17.95 Foundations With COMAL, 2nd Edition, John Kelly (363 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Jr/Sr High level, by Irish professor)
[ ] _____ $28.95/$26.95 Structured Programming With COMAL, Roy Atherton (266 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Intermediate text book, stressing modular programming, High School level, by British professor)
[ ] _____ $20.95/$18.95 Beginning COMAL, Borge Christensen (333 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Beginners text book, Elementary school level, by Danish professor, founder of COMAL)
[ ] _____ $17.95/$15.95 C64 Graphics With COMAL 0.14, Len Lindsay (170 pages)-(shipping add $3)
[ ] _____ $14.95/$4.95 Optional matching disk
(Reference book to COMAL 0.14 Graphics & Sprite commands, companion to COMAL Handbook)
[ ] _____ $6.95/$4.95 COMAL From A To Z, Borge Christensen (64 pages)-(shipping add $2)
(Mini reference book for COMAL 0.14, including its graphics and sprites)
[ ] _____ $14.95/$12.95 Captain COMAL Gets Organized, Len Lindsay (102 pages, with disk)-(ship $2)
(Application tutorial to illustrate benefits of modular programming)
[ ] _____ $6.95/$4.95 COMAL Workbook, Gordon Shigley (69 pages)-(shipping add $2)
(Beginners level - perfect companion to the Tutorial Disk)
[ ] _____ $14.95/$12.95 COMAL Library Functions & Procedures, Kevin Quiggle (71 pgs, with disk)-(ship $2)
(For the 0.14 programmer, 140 procedures and functions ready to merge)
[ ] _____ $14.95/$12.95 Graphics Primer, Mindy Skelton (84 pages, with disk)-(shipping add $2)
(Beginners level tutorial to COMAL 0.14 graphics and sprites)
[ ] _____ $6.95/$4.95 Cartridge Graphics & Sound, Friends of Captain COMAL (64 pages)-(ship add $2)
(Mini reference book to all built in 2.0 cartridge package commands)
[ ] _____ $19.95/$17.95 COMAL 2.0 Packages, Jesse Knight (108 pages, with disk)-(shipping add $2)
(Advanced. For ML programmers- how to write your own packages- with C64sym & supermon)
[ ] _____ $19.95/$17.95 Packages Library, David Stidolph (76 pages, with disk)-(shipping add $2)
(Intermediate. 17 example packages, most with source code on disk- Smooth Scroll Editor too)
[ ] _____ $19.95/$17.95 Cartridge Tutorial Binder, Frank Bason, Leo Hojsholt (320 pgs, with disk)(ship $3)
(Beginners manual to everything in the COMAL 2.0 cartridge)
[ ] _____ $14.95/$12.95 COMAL Quick 0.14 & Utilities Set #2, Jesse Knight (24 pages, with disk)-(ship $2)
(Double sided disk for 0.14, one side FULL of printer utilities, including graphic screen dumps)

OTHER:

[ ] _____ OTHER: _____
[ ] _____ $3.95/$2.95 Keyboard Overlay for C64 COMAL 0.14 - Cheatsheet (shipping add $1)
=====
Total_____ + _____ Shipping = US$_____ **Total Paid (WI add 5%)-(shipping $2 minimum)**
USE OTHER SIDE FOR TOTAL if order includes items on that side
Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Order Form (MORE ITEMS ON OTHER SIDE)

Name:_____ SUBSCRIBER NUMBER:_____ July 1986
(required for reduced prices-except new subs)
Street:_____ Pay by check/MoneyOrder in US Dollars
Canada Postal US Dollar Money Order is OK
City/St/Zip:_____ VISA / MasterCard print card#/exp date:

VISA/MC #:_____exp date:_____Signature:_____

Qnty Price List/Subscriber price-Item Description *(two disks may be supplied as one double sided disk)*
Prices subject to change *(all disks except IBM PC COMAL system are Commodore 1541 format)*

SUBSCRIPTIONS

[ ] _____ **COMAL Today newsletter-> How many? ____ Start with: 6 7 8 9 10 11 12 13 14 <-Circle one**
(each issue is 80 pages of articles, notes, tips, and program listings)(prices go up $4 on Sept 1)
($14.95 for 6; $22.95 for 10 issues; >>> Canada add $1 per issue; >>> overseas add $5 per issue)
[ ] _____ $3.95/$1.50 COMAL Today backissue: circle #s wanted-> 5 6 7 8 9 -(ship add $1 each)
[ ] _____ $3.95/$2.95 COMAL Today backissue: circle #s wanted-> 10 11 12 13 -(ship add $1 each)
[ ] _____ $14.95/$12.95 COMAL Yesterday, first 4 issues of COMAL Today, spiral bound (ship add $2)
[ ] _____ **Today Disk subscription >>> How many disks (at least 6)? ____ Start with disk#_____**
(each disk contains most programs from COMAL Today - plus more! Double sided since #6)
($35.95 for 6; $55.95 for 10 disks---add $5 per disk after first 6---**(no shipping charge)**
[ ] _____ $14.95/$9.75 Today Disk-**Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 11 12 13**
(Disk subscriptions must be 6 or more disks - single Today Disks are available - use above line)

SYSTEMS:

[ ] _____ $98.95/$94.95 Deluxe Cartridge Pak (with the black COMAL 2.0 cartridge)-(shipping add $4)
(also includes: Cartridge Tutorial Binder & disk and Cartridge Graphics & Sound book)
[ ] _____ $74.95/$69.95 Black COMAL 2.0 Cartridge, Plain (no books/no disk)-(shipping add $2)
[ ] _____ **$29.95/$24.95 Super Chip (for black COMAL 2.0 cartridge empty socket) includes C128 support**
[ ] _____ **$5/$5 Install and test Super Chip fee. Done for no charge if cartridge purchased at same time.**
[ ] _____ $19.95/$17.95 Programmers Paradise Pak (includes COMAL Today 5,6,7,8,9 & 10)(ship add $2)
(includes Fastloaded COMAL 0.14 system & COMAL From A To Z book)
[ ] _____ **$5/$5 option - only with Paradise Pak** - (includes Tutorial disk, Best of Disk, Auto Run Demos)
[ ] _____ **$29.95/27.95 COMAL 0.14 Starter Kit (5 disks, 2 books, 6 newsletters, more)-(ship add $4)**
(includes everything in Paradise Pak, plus $5 option, plus Graphics Primer disk/book)
[ ] _____ $350/$350 IBM Denmark PC COMAL 2.0 (Danish manual/COMAL Handbook)-(shipping add $5)
(system is in English, works on most IBM PC Compatibles with at least 256K)

DISKS:

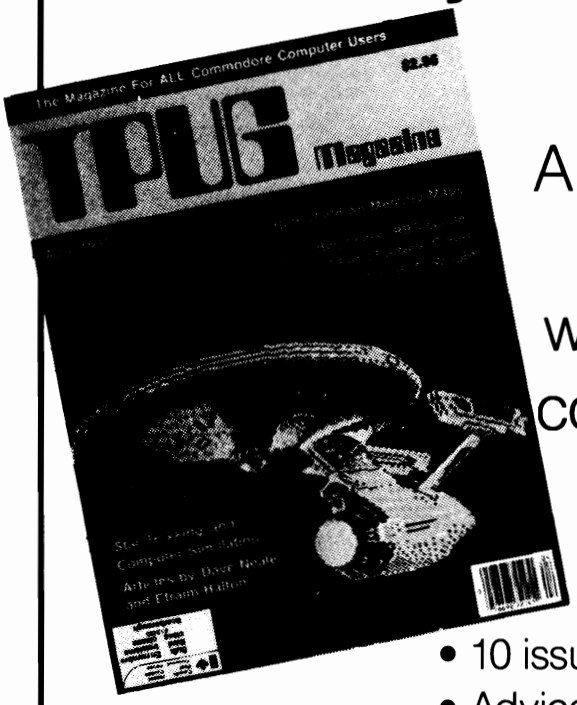
[ ] _____ **$14.95/$9.75 Read & Run disk (for 0.14 & 2.0)**
[ ] _____ **$14.95/$9.75 Shareware disk (for 2.0) -- Buy #1 -- Get #2 FREE. (Due July/Aug 86)**
[ ] _____ $14.95/$9.75 Best of COMAL 0.14 (new version - single side of disk)
[ ] _____ $14.95/$9.75 Auto RUN Demo and Tutorial Disk (perfect with COMAL Workbook)
[ ] _____ $14.95/$9.75 Bricks Tutorials (2 sided beginners disk - an expanded Tutorial disk)
[ ] _____ $14.95/$9.75 Utility Disk #1 for COMAL 0.14 (Utility Disk #2 see books section)
[ ] _____ $10/$9.75 User Group Disks 0.14 - Circle disks wanted: 1 2 3 4 5 6 7 8 9 10 12
[ ] _____ $10/$9.75 User Group Disks 2.0 - Circle disks wanted: 11 13 14 15
[ ] _____ $29.95/$14.95 Set of all Cartridge Demo Disks (includes #1 #2 #3 & #4)
[ ] _____ $14.95/$9.75 Single Cartridge Demo Disk, Circle one wanted --> 1 2 3 4
[ ] _____ $14.95/$9.75 Slide Show Picture Disks - both #1 & #2 - (HRG type pictures -- 0.14 system)
[ ] _____ $14.95/$9.75 Games Disk #1 (for 0.14 & 2.0)
[ ] _____ $14.95/$9.75 Typing Disk (for 2.0 only)
[ ] _____ $14.95/$9.75 Modem Disk (for 0.14 & 2.0)
[ ] _____ $14.95/$9.75 Font Disk (for 0.14 & 2.0) - dozens of fonts (compatible with PaperClip too!)
[ ] _____ $94.05/$89.95 19 Disk Set (about 1000 programs for COMAL 0.14)-(shipping add $3)

=====
Total_____ + _____ Shipping (this side)
Total_____ + _____ Shipping (other side)
===== + =====
Total_____ + _____ = US$_____ Total Paid (WI add 5%)-(shipping $2 minimum)

Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

TPUG

MORE THAN JUST ANOTHER COMPUTER MAGAZINE!



A membership in the world's largest computer club will provide you with:

- 10 issues of TPUG magazine
- Advice from experts like Jim Butterfield and Elizabeth Deal
- Access to our huge library of public domain software
- Access to our online help services
- All for only $25/yr.

TPUG
101 Duncan Mill
Suite G7
Don Mills, Ontario
CANADA
M3B 1Z3

JOIN US NOW!!

I want to join TPUG as an associate member.

Name _____ Home phone (____) _____

Address _____ Work phone (____) _____

City/Town _____ ☐ Cheque enclosed

Prov/State _____ ☐ Money order enclosed

Postal/Zip Code _____ Country _____ ☐ Visa ☐ MasterCard

My computer equipment is: _____ Card# _____

_____ Expiry _____

_____ Signature _____

Expand Past Maximum Capacity!

The Transactor

The Tech/News Journal For Commodore Computers

At better book stores everywhere! Or 6 issues delivered to your door for just $15.00
That's 29% off the newstand price! (Overseas $21 U.S. Air Mail $40 U.S.)
The Transactor. 500 Steeles Ave. Milton, Ontario. L9T 3P7. 416 878-8438
Also check out The Transactor Disk; every program from each issue, in order as they appear
and The Complete Commodore Inner Space Anthology; over 2.5 million characters of reference information exclusively.
Included are memory maps for BASIC, COMAL, CBM disk drives, BBS numbers, machine language charts, Kernel routine summaries, printer commands, recording formats, I/O port maps, port pinouts, audio control tables, video reference charts, keyword values, commands for common software packages, MLM and assembler commands, and there's more!
To us, expansion knows no limits!